Numerical solution of ODEs

Arne Morten Kvarving

Department of Mathematical Sciences
Norwegian University of Science and Technology

November 5 2007

Problem and solution strategy

- We want to find an approximation to the solution of

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y(t))$$
$$y(t_0) = y_0.$$

- I usually name the independent variable t, while Kreyzig uses x. The reason is that I find it more intuitive - we call this kind of problem an *initial value problem*.
- We know that the solution is given by

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\,\mathrm{d}s.$$

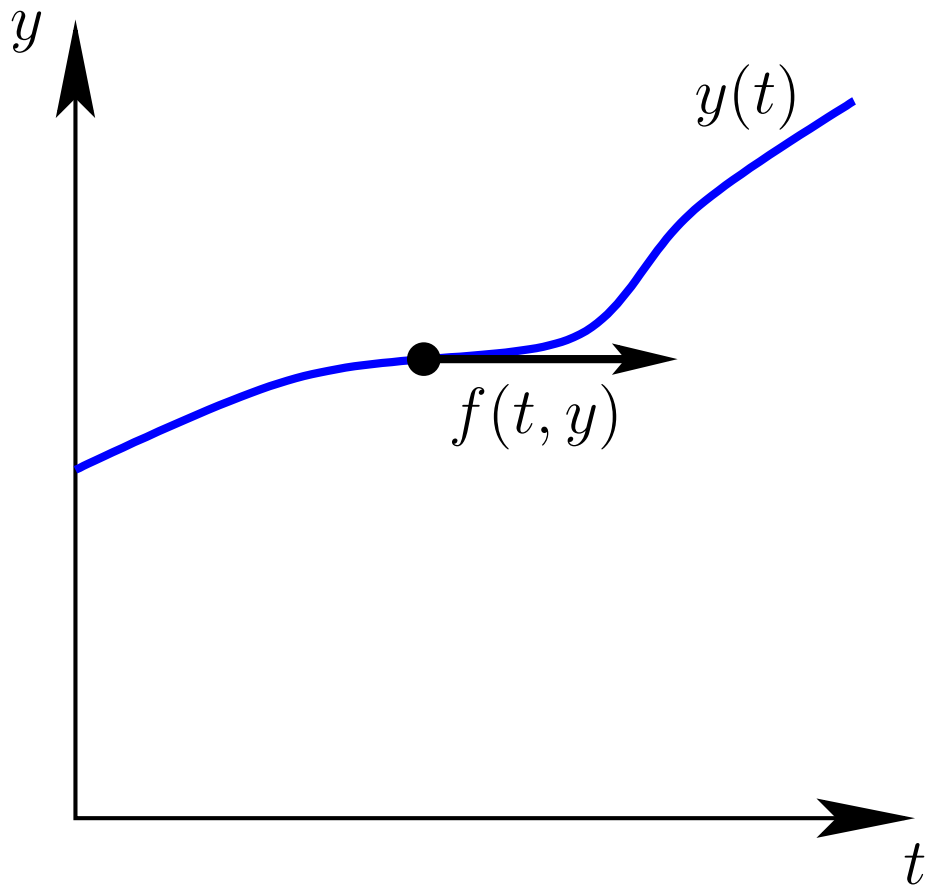
- It is tempting to try the numerical integration rules we have considered earlier. However, these are only valid for autonomous systems:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y(t)),\ y(t_0) = y_0.$$

Problem and solution strategy

- There are two main classes of solution methods for ODEs: First one is one step methods (named so because we only use the value of y at one time level to approximate the value at the next level.
- The other class is multistep methods. These use the value of y at several time levels to approximate the solution at the next level. These methods are not part of the curriculum.
- Additionally, the two classes of methods can each be divided into two parts again:
  - Explicit methods. These are methods where we only use known data to approximate the solution at the next time level.
  - Implicit methods. These are methods where we have to solve a system of equations at each time level. In general the system of equations will be nonlinear and we have to use a method for solving nonlinear systems of equations - e.g. Newton's method. This makes these methods very expensive to use, however in some cases they still are the best choice - for *stiff equations*.

Euler's method



At each point the differential equation gives the direction of the solution.

Euler's method

- Naive method: In each point f describes the tangent of the solution.
- We do a step of size h along this tangent.
- This gives the method

$$y_{n+1} = y_n + hf(t_n, y_n).$$

  It is known as Euler's method.
- We can also motivate the method using Tayler expansion:

$$\begin{aligned} y(t_{n+1}) &= y_n + hy'_n + \frac{h^2}{2}y''_n + \mathcal{O}\left(h^3\right) \\ &= y_n + hf(y_n) + \frac{h^2}{2}y''_n(t_n) + \mathcal{O}\left(h^3\right). \end{aligned}$$

Euler's method

- From the Taylor expansion we see that in each step we do an error which is $\mathcal{O}\left(h^2\right)$. This is known as the *local error*: Assume that we do one step starting with exact value $y_n = y(t_n)$,

$$|y(t_n + h) - y_{n+1}| = \frac{h^2}{2} y''(t_n) + \mathcal{O}\left(h^3\right) = \mathcal{O}\left(h^2\right)$$

- This error will propagate since we take many steps.
- One can show that this propagation gives us one order lower in the *global error*. Assume that we start we $y_0 = y(t_0)$ and do n steps of size h:

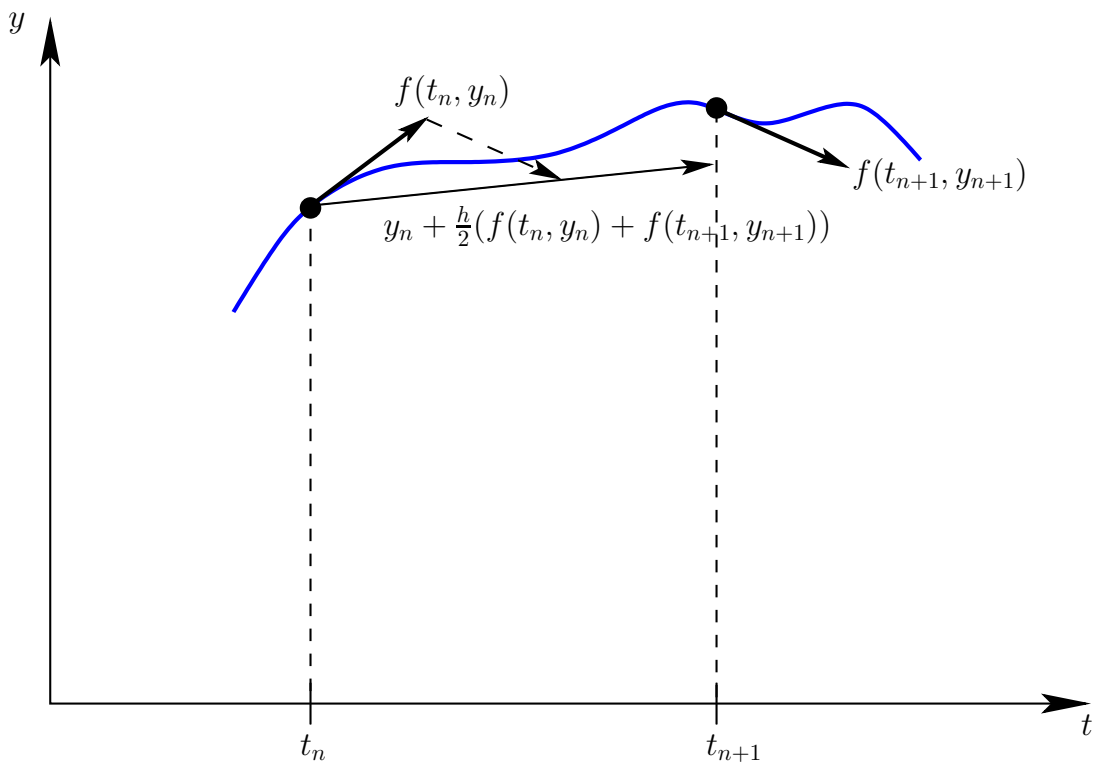
$$|y(t_0 + nh) - y_n| = \mathcal{O}\left(h\right)$$

This means that Euler's method is a first order method.

Euler's method

- Coincides with the rectangular rule for numerical integration.
- Erwin Kreyzig claims that this method is "never used in real computations".
- This is not true. We sometimes have to use it, for instance when evaluating f is very costly.
- "Preferably avoided" is a better way to put it.
- It is often the starting point when we want to construct better methods.

Heun's method: Improved Euler



We use the mean of two tangents.

Heun's method: Improved Euler

- Euler only utilizes the tangent in the starting point of the interval.
- If you remember the trapezoidal rule, what we did was that we use the mean of the tangents at the starting point and at the ending point of the subinterval.
- This yields the trapezoidal rule for ODEs:

$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(t_n, y_n\right) + f\left(t_{n+1}, y_{n+1}\right)\right)$$

  As you can see, this is an example of an implicit method.
- What happens if we replace $f(t_{n+1}, y_{n+1})$ with $f(t_{n+1}, y_n + hf(t_n, y_n))$? That is, we approximate the solution at the end point using a normal Euler step.

Heun's method: Improved Euler

- This yields Heun's method:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_{n+1}, y_n + hk_1) \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2) \end{aligned}$$

  Note the particular form we write the method on - you will understand why later.

- We consider the Taylor expansion:

$$\begin{aligned} |y(t_n + h) - y_n| &= hy'(t_n) + \frac{h^2}{2}y''(t_n) + \mathcal{O}\left(h^3\right) \\ &= hf(y_n) + \frac{h^2}{2}f'(y_n) + \mathcal{O}\left(h^3\right) \end{aligned}$$

Heun's method: Improved Euler

- The derivative can be approximated by

$$f'(t_n, y_n) = \frac{f(y_{n+1}) - f(y_n)}{h} + \mathcal{O}(h)$$
$$\approx \frac{f(y_n + hf(y_n)) - f(y_n)}{h} + \mathcal{O}(h)$$

  The error we make by approximating $f(y_{n+1})$ using an Euler step is, as stated earlier, $\mathcal{O}(h^2)$. Due to the h in the denominator this error will also be $\mathcal{O}(h)$ and hence it is included in the term $\mathcal{O}(h)$.
- We now insert this into the Tayler expansion and find

$$|y(t_n + h) - y_n| = hf(y_n)$$
$$+ \frac{h^2}{2}\left(\frac{f(y_n + hf(y_n)) - f(y_n)}{h}\right)$$
$$+ \frac{h^2}{2}\mathcal{O}(h) + \mathcal{O}(h^3).$$

Heun's method: Improved Euler

- From this we deduce that Heun's method has a local error of order 3, and hence the global error is of order 2 - Heun's method is a second order method.
- The method can also be stated in the form

$$y^*_{n+1} = y_n + hf(t_n, y_n)$$
$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(t_n, y_n\right) + f\left(t_{n+1}, y^*_{n+1}\right)\right)$$

  When stated on this form we see that the method is a so called *predictor-corrector*-method.
- We first "predict" a solution in the first step, then we "correct" this prediction in the second step.

Runge-Kutta methods

- The change we did from Euler $\longrightarrow$ Heun can be systemized.
- We calculate $k_1, k_2, k_3, \cdots, k_s$-values within the interval. These are calculated using methods with lower order.
- Finally we use a weighted sum of these k's.
- In general the methods can be put in the form

$$k_r = f\left(t_n + c_r h, y_n + h\sum_{j=1}^{r} a_{rj} k_j\right), \quad r = 1, 2, 3, \cdots, s$$

$$y_{n+1} = y_n + h\sum_{r=1}^{s} b_r k_r$$

We call s the number of *stages* in the method.

Runge-Kutta methods

- There are infinitely many methods on this form. We can construct methods of arbitrary order (finding methods with high order is far from trivial though), and these methods are OFTEN used in real life applications.
- These methods can be classified by
  - the matrix $\underline{A}$ (how much shall we weigh each k-value at each stage?).
  - the vector $\underline{c}$ (at which time level is the stage an approximation of the tangent?)
  - the vector $\underline{b}$ (how much shall we weigh each k-value in the final update?).
- This information is often presented in a *RK-tableaux*, which is of the form

$$\begin{array}{c|c} \underline{c} & \underline{A} \\ \hline & \\ & \underline{b}^T \end{array}$$

Runge-Kutta methods

- Here we only consider explicit methods. This means that
  - The first stage will always consist of evaluating f at the starting point - it is the only information we have at hand.
  - The matrix $\underline{A}$ must be strictly lower triangular (no diagonal elements).
- This means that our tableaux is of the form

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & \cdots \\ c_2 & a_{21} & 0 & \vdots & \vdots \\ c_3 & a_{31} & a_{32} & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ \hline & b_1 & b_2 & b_3 & \cdots \end{array}$$

Runge-Kutta methods

- Inserting this info into the general form our methods can be stated as

$$k_1 = f\left(t_n, y_n\right)$$

$$k_r = f\left(t_n + c_r h, y_n + h\sum_{j=1}^{r-1} a_{rj} k_j\right), \quad r = 2, 3, \cdots, s$$

$$y_{n+1} = y_n + h\sum_{r=1}^{s} b_r k_r$$

Runge-Kutta methods

- Let us find the tableaux for Heun's method, that is the method

$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_{n+1}, y_n + hk_1)$$
$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2).$$

- We have $s = 2$ stages. We have $c_1 = 0$ (since it is an explicit method), $c_2 = 1$, $b_1 = \frac{1}{2}$ and $b_2 = \frac{1}{2}$.
- Again, since this is an explicit two-stage method, we have only one coefficient different from zero in the matrix $\underline{A}$, namely $a_{21} = 1$. Together this gives the tableaux

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

Runge-Kutta methods

- Here we consider a much used method, ERK4.
- As the name indicate this is an explicit Runge-Kutta method of order 4.
- The tableaux is given by

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

Runge-Kutta methods

- Utilizing the coefficients given, we can explicitly state the method as

$$
\begin{aligned}
k_1 &= f\left(t_n, y_n\right) \\
k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\
k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\
k_4 &= f\left(t_n + h, y_n + hk_3\right) \\
y_{n+1} &= y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned}
$$

- This method concides with Simpson's method for autonomous systems.

Error control - adaptive solvers

- It is of great interest to be able to control how large an error we make.
- As motivation you can consider the same scenario as we had in adaptive numerical integration: We would like to use small h where the solution varies alot, while using a larger h where the solution varies less.
- Two approaches:
  - As in numerical integration - use different values for h.
  - Use two different methods of different order.

Error control - varying the step size

- We apply the same method twice, first with a stepsize $2h$, then twice with a step size h.
- As an example, consider ERK4.
  - First we use step size $2h$ to obtain the approximation $\tilde{y}$. This yields

    $$y(2h) - \tilde{y} = \frac{1}{2} C (2h)^5 = 16\epsilon$$

    That is, we have $\epsilon = Ch^5$. The factor $\frac{1}{2}$ is there as we will do half the amount of steps using step size $2h$.
  - We then do two steps with step size h to obtain $\tilde{\tilde{y}}$. This yields

    $$y(2h) - \tilde{\tilde{y}} = \epsilon$$

  - We now have two expressions for $y(2h)$ and we find that

    $$y(2h) = \tilde{y} + 16\epsilon = \tilde{\tilde{y}} + \epsilon \Rightarrow$$
    $$\epsilon = \frac{1}{15} (\tilde{\tilde{y}} - \tilde{y})$$

Error control - varying the step size

- This approach is very expensive to use in real life computations. Why? Because we do a considerable amount of extra work just to be able to estimate the error done in a step.
- It is of interest to find methods where the error estimate comes approximately "free", that is, methods that lets us avoid doing much extra work just to estimate the error.
- One way to do this is by varying the order of the method instead of the step size.

Error control - varying the order

- Example: RKF45

$$\text{RKF4} : \tilde{y} = y(h) + \mathcal{O}\left(h^5\right)$$
$$\text{RKF5} : \tilde{\tilde{y}} = y(h) + \mathcal{O}\left(h^6\right)$$
$$\tilde{\tilde{y}} - \tilde{y} \approx Ch^5.$$

  We assume that $Ch^6 << Ch^5$. Hence we have an error estimate for solution obtained using $RKF4$.

- We can combine many methods in this way, for example Euler-Heun. The important part is that we can evaluate the method of higher order cheap once we have done the calculations needed for the lower order method. For Runge-Kutta methods this means that we are searching for methods which employs the same k values, that is, the only difference between the two methods should be the vector $\underline{b}$.

Summary - explicit methods

Method	Function evaluations	Global error	Local error
Euler	1	1	2
Heun	2	2	3
RK4	4	4	5
RKF4	5	4	5
RKF5	6	5	6
RKF	6	4	5

Note that RKF4 is not a good method for "strict" 4. order since we use 5 function evaluations.

Summary - explicit methods

- All the methods we have considered works nicely for systems of first order ODEs.
- The formulas are exactly the same, the only difference is that we have to do everything on vectorial form.
- If we are given a scalar m'th order ODE

$$y^{(m)} = f\left(t, y, y', y'', \cdots, y^{(m-1)}\right),$$

  we can state it as a system of first order ODEs in order to be able to apply our methods.
- Hence there has been very little focus on methods for higher order ODEs. One exception is the Runge-Kutta-Nyström schemes.

How to make a system out of a higher order ODE

- We are given a scalar m'th order ODE

$$y^{(m)} = f\left(t, y, y', y'', \cdots, y^{(m-1)}\right).$$

- We introduce vector components

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_m &= y^{(m-1)}. \end{aligned}$$

This is basically just a renaming of the variables involved.

How to make a system out of a higher order ODE

- If we now insert this into our problem, we get a system of ODEs:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_m' &= f\left(t, y_1, y_2, y_3, \cdots, y_m\right) \end{aligned}$$

- One application of this of particular interest: Introduce

$$\begin{aligned} y_1 &= t \\ y_2 &= y. \end{aligned}$$

This allows us to state our equation in autonomous form as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}' = \begin{pmatrix} 1 \\ f\left(\underline{y}\right) \end{pmatrix}.$$

Runge-Kutta-Nyström methods

- Consider the equation

$$y'' = f\left(t, y, y'\right)$$

- We apply our trick to state it as a system of first order ODEs,

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= f\left(t, y_1, y_2\right). \end{aligned}$$

- Now, assume that f does not depend on y'. It then turns out that if we apply (particular) RK methods to the system, two or more stages will coincide, and hence the methods will be cheaper. These methods are known as RKN methods.

Why consider implicit methods - stiff equations

- Consider the equation (in this context known as the Dahlquists test equation)

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \lambda y$$
$$y(0) = y_0$$

- We know that the exact solution is given by

$$y(t) = y_0 e^{\lambda t}.$$

  This solution exists for $t \to \infty$ if and only if $\mathrm{Re}\ \lambda < 0$. This behaviour of the solution is something we would like our numerical solution to reflect.
- What happens if we apply Euler's method to this equation?

$$y_{n+1} = y_n + hf(t_n, y_n) = y_n + h\lambda y_n = (1 + h\lambda)\, y_n$$

  Applying this recursively we obtain

$$y_n = (1 + h\lambda)^n\, y_0.$$

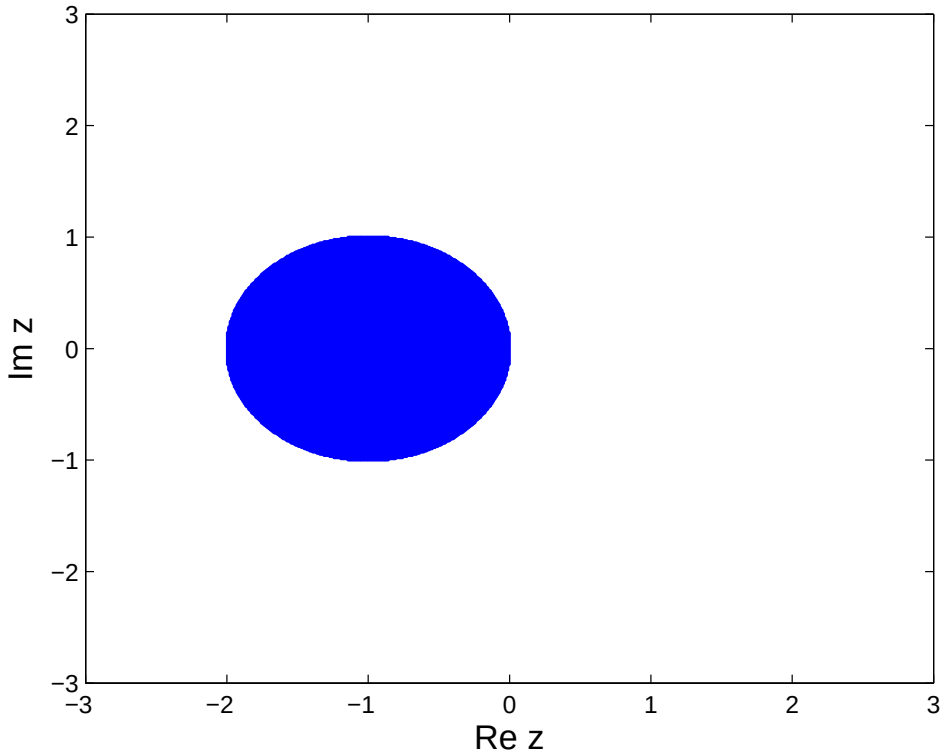
Why consider implicit methods - stiff equations

- Hence, if the method should yield a solution for $t \to \infty \Leftrightarrow n \to \infty$ we need

$$|1 + h\lambda| < 1.$$

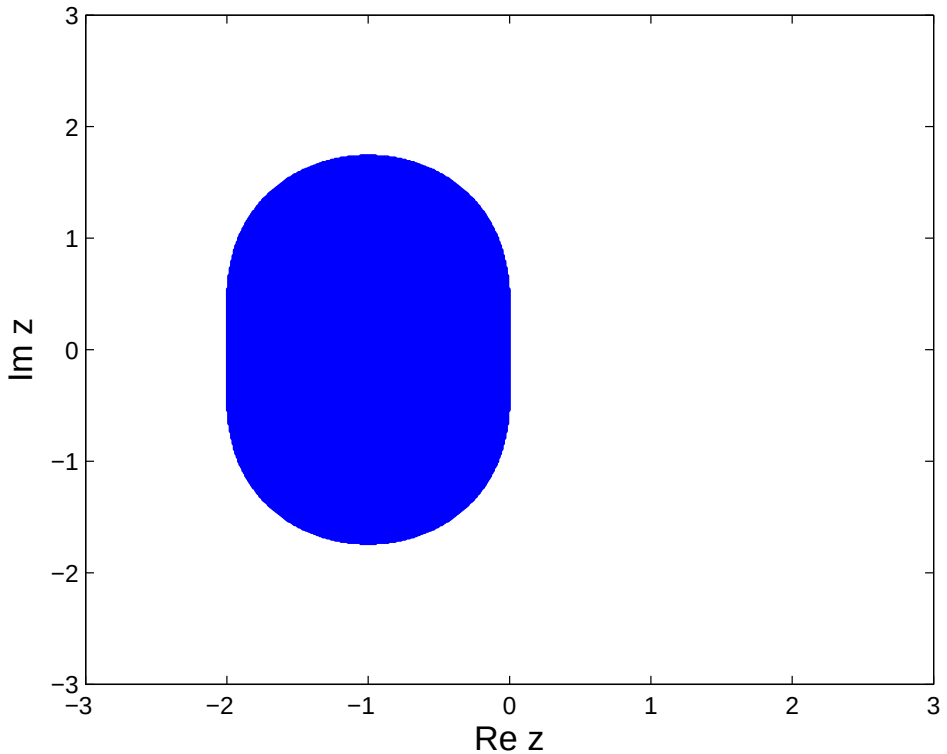
- We now let $z = h\lambda$. The area where we get a stable solution is plotted in the Figure beneath. Remember, the exact solution is damped for all $\lambda \in \mathbb{C}^-$ while the method only dampens the solution in the filled area.

Why consider implicit methods - stiff equations



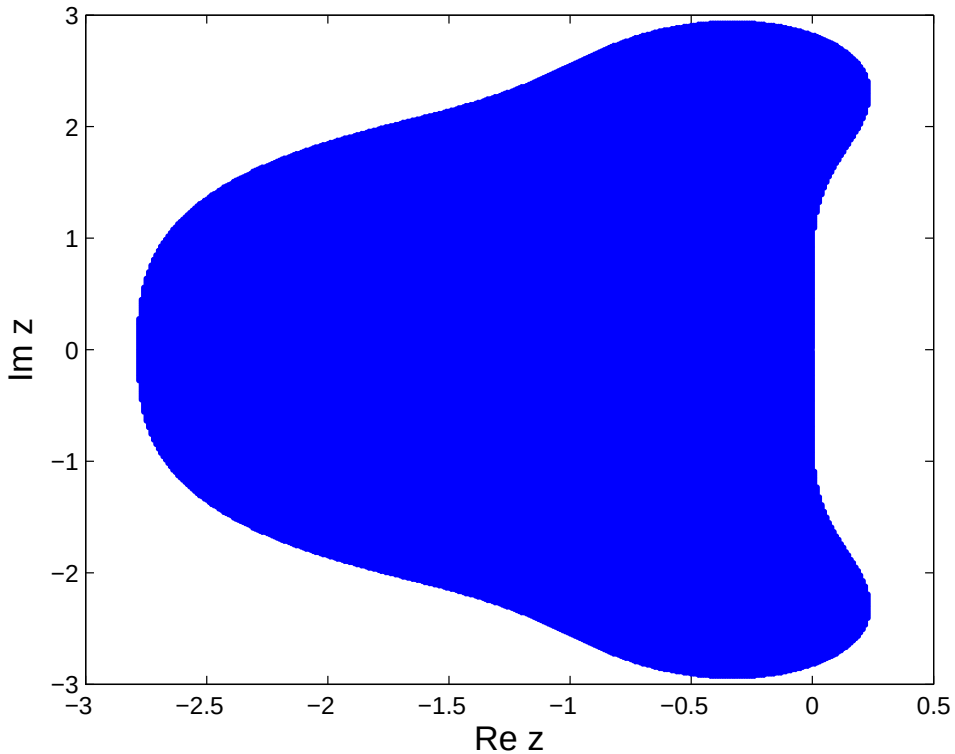
Stability area for Euler's method.

Why consider implicit methods - stiff equations



Stability area for Heun's method.

Why consider implicit methods - stiff equations



Stability area for explicit RK4.

Why consider implicit methods - stiff equations

- This means that if λ is large, we have to use a very small h to stay within the stability area.
- So how does this relate to general equations? Let us consider a system of equations of the form

$$\frac{\mathrm{d}\underline{y}}{\mathrm{d}t} = \underline{A}\,\underline{y}$$

and that the matrix $\underline{A}$ can be diagonalized. That is, we can find $\underline{Q}$ and $\underline{\Lambda}$ such that

$$\underline{Q}\,\underline{\Lambda}\,\underline{Q}^{T} = \underline{A}.$$

where $\underline{Q}$ is an orthogonal matrix with the eigenvectors of $\underline{A}$ as columns and $\underline{\Lambda}$ an diagonal matrix with the eigenvalues of $\underline{A}$ along the diagonal.

Why consider implicit methods - stiff equations

- We now insert this into the equation to obtain

$$\begin{aligned}\frac{\mathrm{d}\underline{y}}{\mathrm{d}t} &= \underline{A}\,\underline{y} \\ &= \underline{Q}\,\underline{\Lambda}\,\underline{Q}^{T}\underline{y}\end{aligned}$$

Since $\underline{Q}$ is orthogonal, its inverse is given by $\underline{Q}^{-1} = \underline{Q}^{T}$. We multiply with this from the left:

$$\begin{aligned}\underline{Q}^{T}\frac{\mathrm{d}\underline{y}}{\mathrm{d}t} &= \underline{\Lambda}\,\underline{Q}^{T}\underline{y} \Rightarrow \\ \frac{\mathrm{d}\underline{z}}{\mathrm{d}t} &= \underline{\Lambda}\,\underline{z}\end{aligned}$$

where we have set $\underline{z} = \underline{Q}^{T}\underline{y}$.

Why consider implicit methods - stiff equations

- Hence we have one equation of Dahlquist type along each eigenvalue of the matrix $\underline{A}$ (remember, $\underline{\Lambda}$ is diagonal and hence there are no coupling between the equations).
- We need to stay within the stability area for each eigenvalue - which means that the largest eigenvalue dictates the maximum step size we can use.
- If $\underline{A}$ has one eigenvalue with a large amplitude, the equation is said to be *stiff*.

Why consider implicit methods - stiff equations

- Now, what happens for an implicit method? Let us consider the trapezoidal rule applied to the test equation:

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

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

$$y_n = \left(\frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}}\right)^n y_0$$

- The area where this gives a stable solution includes the entire left half plane. That is, the method gives a stable solution wherever the exact solution exists. This property of a method is called *A-stability*, and one can show that no explicit method can be A-stable. This means that we have *no* restriction on the step size.

Why consider implicit methods - summary

- For some equations the restrictions on the step size we need to honor if we use an explicit method is so strict that the methods are practically useless in real life applications.
- We then resort to implicit methods. Even if we have to solve a system of equations for each step, the total algorithm is still cheaper since we can choose step size based on accuracy considerations only.