AMSC/CMSC 660 Scientific Computing I

Fall 2008

UNIT 5: Numerical Solution of Ordinary Differential Equations

Part 1

Dianne P. O'Leary

The Plan

- Initial value problems (IVPs) for ordinary differential equations (ODEs)
  - Review of undergraduate material
  - Hamiltonian systems
- Differential-Algebraic Equations
  - Some basics
  - Some numerical methods
- Boundary value problems for ODEs.
  - Some basics
  - Shooting methods
  - Finite difference methods

References:

These notes are based on Unit V in the textbook.

Initial value problems for ordinary differential equations

- Review of undergraduate material
- Hamiltonian systems

Review of undergraduate material

Numerical Solution of Ordinary Differential Equations

Jargon: ODE = ordinary differential equation

Notation:

- $y_{(i)}$ will denote the ith component of the vector $\mathbf{y}$.
- y_i will denote our approximation to the function y evaluated at t_i.

The plan

- ODEs: manipulation and properties
  - Standard form
  - Solution families
  - Stability
- ODEs: numerical methods
  - Using the software
  - Euler's method
  - The backward Euler method
  - The Adams family
  - Building a practical ode solver
  - Runge-Kutta methods

ODEs: Manipulation and properties

(What we need to know from your previous MATH class)

- Standard form
- Solution families
- Stability

Standard form of the ODE

We'll only work with problems in standard form, because that is what the software needs:

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(0) &= \mathbf{y}_0 \end{aligned}$$

where the function $\mathbf{y}$ has m components, $\mathbf{y}'$ means the derivative with respect to t, and $\mathbf{y}_0$ is a given vector of numbers. Writing this component-by-component yields

$$\begin{aligned} y'_{(1)} &= f_1(t, y_{(1)}, \ldots, y_{(m)}) \\ &\cdots \\ y'_{(m)} &= f_m(t, y_{(1)}, \ldots, y_{(m)}) \end{aligned}$$

with $y_{(1)}(0), \ldots, y_{(m)}(0)$ given numbers.

Note: It is not essential that we start at $t = 0$; any value will do.

Writing problems in standard form

Example: Volterra's model of rabbits (with an infinite food supply) and foxes (feeding on the rabbits):

$$\begin{aligned} \frac{dr}{dt} &= 2r - \alpha r f \\ \frac{df}{dt} &= -f + \alpha r f \\ r(0) &= r_0 \\ f(0) &= f_0 \end{aligned}$$

The parameter α is the encounter factor, with $\alpha = 0$ meaning no interaction.

Let $y_{(1)} = r$ and $y_{(2)} = f$. Then we can write this in standard form.

Example: a second order equation

$$\begin{aligned} u'' &= g(t, u, u') \\ u(0) &= u_0 \\ u'(0) &= v_0 \end{aligned}$$

where u_0 and v_0 are given numbers.

Let $y_{(1)} = u$ and $y_{(2)} = u'$. Then we can write this in standard form.

Note:

- Every integration problem can be written as an ODE, but the converse is not true.

  Example:

  $$\int_a^b f(t)dt$$

  is equal to $y(b)$, where

  $$\begin{aligned} y' &= f(t) \\ y(a) &= 0 \end{aligned}$$

Solution families

Given $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, the family of solutions is the set of all functions $\mathbf{y}$ that satisfy this equation.

Three examples of solution families

Example 1: parallel solutions

$$y' = e^{-t}$$

has the solutions

$$y(t) = c - e^{-t}$$

where c is an arbitrary constant.

Example 2: solutions that converge

$$y' = -y$$

has the solution

$$y(t) = ce^{-t}$$

where c is an arbitrary constant.

Example 3: solutions that diverge

$$y' = y$$

has the solution

$$y(t) = ce^{t}$$

where c is an arbitrary constant.

Stability of ODEs

Loosely speaking,

- An ODE is stable if family members move toward each other as t increases. See example 2.
- An ODE is unstable if family members move apart as t increases. See example 3.
- An ODE is on the stability boundary if family members stay parallel. See example 1.

More precisely, let

$$f_y = \partial f / \partial y.$$

Then a single ODE is

- stable at a point $\hat{t}, \hat{y}$ if $f_y(\hat{t}, \hat{y}) < 0$.
- unstable at a point $\hat{t}, \hat{y}$ if $f_y(\hat{t}, \hat{y}) > 0$.
- stiff at a point $\hat{t}, \hat{y}$ if $f_y(\hat{t}, \hat{y}) << 0$.

A system of ODEs is

- stable at a point $\hat{t}, \hat{\mathbf{y}}$ if the real part of all the eigenvalues of the matrix $\mathbf{f}_y(\hat{t}, \hat{\mathbf{y}})$ are negative.
- stiff at a point $\hat{t}, \hat{\mathbf{y}}$ if the real parts of more than one eigenvalue of $\mathbf{f}_y(\hat{t}, \hat{\mathbf{y}})$ are negative and wildly different.

Stability examples for a single equation:

- For $y' = e^{-t}$, $f_y = 0$.
- For $y' = -y$, $f_y = -1$.
- For $y' = y$, $f_y = 1$.

Stability examples for a system of equations:

$$\mathbf{y}' = \mathbf{A}\mathbf{y} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \mathbf{y}$$

The solution is

$$\mathbf{y}(t) = b_1 e^{\lambda_1 t}\mathbf{z}_1 + b_2 e^{\lambda_2 t}\mathbf{z}_2$$

where the λs and $\mathbf{z}$s are eigenvalues and eigenvectors of $\mathbf{A}$:

$$\begin{aligned} \mathbf{A}\mathbf{z}_1 &= \lambda_1 \mathbf{z}_1 \\ \mathbf{A}\mathbf{z}_2 &= \lambda_2 \mathbf{z}_2 \end{aligned}$$

Let

$$\mathbf{A} = \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix}.$$

Then

$$\begin{aligned} \mathbf{A}\begin{bmatrix} 2 \\ -1 \end{bmatrix} &= -\begin{bmatrix} 2 \\ -1 \end{bmatrix} \\ \mathbf{A}\begin{bmatrix} -1 \\ 1 \end{bmatrix} &= -1000\begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{aligned}$$

So this is an example of a stiff system. The solution is

$$\mathbf{y} = b_1 e^{-t}\begin{bmatrix} 2 \\ -1 \end{bmatrix} + b_2 e^{-1000t}\begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

If $y_{(1)}(0) = y_{(2)}(0) = 1$, then

$$b_1 = 2, \;\; b_2 = 3.$$

Numerical Methods for ODEs

- Using the software
- Euler's method
- Backward Euler method
- The Adams family
- Building a practical ode solver
- Runge Kutta methods

Using the software

Suppose we want to solve the foxes-and-rabbits model:

$$\begin{aligned} \frac{dr}{dt} &= 2r - \alpha r f \\ \frac{df}{dt} &= -f + \alpha r f \\ r(0) &= r_0 \\ f(0) &= f_0 \end{aligned}$$

We'll use the Matlab function ode45, which is good for nonstiff systems of equations. (Our problem is actually stiff for some values of α, but this is just an example of how the calling sequence goes.)

Here is the first part of the documentation, showing the simplest way to use the software:

```
ODE45  Solve non-stiff differential equations, medium order method.
   [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL]
   integrates the system of differential equations y' = f(t,y)
   from time T0 to TFINAL with initial conditions Y0. ODEFUN is a
   function handle. For a scalar T and a vector Y, ODEFUN(T,Y)
   must return a column vector corresponding to f(t,y). Each row
```

```
    in the solution array YOUT corresponds to a time returned in
    the column vector TOUT.  To obtain solutions at specific
    times T0,T1,...,TFINAL (all increasing or all decreasing),
    use TSPAN = [T0 T1 ... TFINAL].
```

So if we want to find how many rabbits and foxes there are at times $0, .1, \ldots, 1.9, 2$, if there are initially 20 rabbits and 10 foxes, we say

```
[t,y] = ode45(@rabfox,[0:.1:2], [20,10]);
```

and we supply the function `rabfox` as follows:

```
function f = rabfox(t,y)
% Computes y' for the Volterra model.
% y(1) is the number of rabbits at time t.
% y(2) is the number of foxes at time t.

global alpha % interaction constant

t  % a print statement, just so we can see how fast
   % the progress is, and what stepsize is being used

f(1,1) = 2*y(1) - alpha*y(1)*y(2);
f(2,1) =  -y(2) + alpha*y(1)*y(2);
```

Our main program, to try various values of α, looks like this:

```
% Run the rabbit-fox model for various values of
% the encounter parameter alpha, plotting each
% solution.

global alpha

for i=2:-1:0,
   alpha = 10^(-i)
[t,y] = ode45(@rabfox,[0:.1:2], [20,10]);
plot(t,y(:,1),'r',t,y(:,2),'b');
   legend('rabbits','foxes')
   title(sprintf('alpha = %f',alpha));
   pause
end
```

The ODE solvers can do more complicated things for you, too; read the documentation carefully.

Once we learn about solving nonlinear equations, we'll be able to solve problems like this: Suppose we want to find a value of α so that the final number of rabbits is equal to 4:

$$r_\alpha(2) - 4 = 0.$$

Note that our experiment above tells us that there is such a value of α between .01 and 1.

Write a program that uses `ode45` along with the nonlinear equation solver `fzero` to find α. []

Now we'll turn our attention to what is inside a code like `ode45`. We'll start with simpler algorithms, and work our way up.

Note: I'll write the formulas for a single differential equation, but the change to a system of equations will just mean changing y to $\mathbf{y}$ and f to $\mathbf{f}$.

Euler's method

We use Euler's method to illustrate the basic ideas behind numerical methods for ODEs. It is not a practical method because it is too slow, but it will help us understand the ideas.

Three derivations of Euler's method

Suppose we know t_n, $y_n = y(t_n)$, and $f_n = f(t_n, y_n)$.

Approach 1: Geometry picture

Approach 2: Polynomial interpolation

Unquiz: Find the linear polynomial that interpolates the given data, and then evaluate it at t_{n+1}.

Approach 3: Taylor series

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(\xi)$$

for some point $t \le \xi \le t+h$. Note that

$$y'(t) = f(t, y(t))\,.$$

This gives the formula

$$y_{n+1} = y_n + hf_n\,.$$

How to use Euler's method

Given y_0 and $t_0, t_1, \ldots t_N$,

For $n = 0, \ldots, N-1$,

$$y_{n+1} = y_n + (t_{n+1} - t_n) f(t_n, y_n)$$

Unquiz: Apply this to $y' = 1$, $y(0) = 0$, using a stepsize of $h = .1$ []

Sources of error

- round-off error, especially if the steps get too small.
- local error: the error assuming that y_n is the true value.
- global error: how far we have strayed from our original solution curve.

Local error

Taylor series tells us that

$$y(t_{n+1}) - y_{n+1} = \frac{h^2}{2} y''(\xi)$$

Global error

We know

$$y(t_{n+1}) = y(t_n) + h_n f(t_n, y(t_n)) + \frac{h^2}{2} y''(\xi_n), \quad n = 0, 1, 2, \ldots,$$

and

$$y_{n+1} = y_n + h_n f(t_n, y_n)\,.$$

Therefore,

$$y(t_{n+1}) - y_{n+1} = y(t_n) - y_n + h_n(f(t_n, y(t_n)) - f(t_n, y_n)) + \frac{h^2}{2} y''(\xi_n)\,.$$

- The plum-colored terms are global errors.
- The black term is the local error.
- Using the MVT, the blue term can be written as

$$h_n(f(t_n, y(t_n)) - f(t_n, y_n)) = h_n f_y(\eta)(y(t_n) - y_n)\,.$$

So we have the expression

$$\text{new global error} = (1 + h_n f_y(\eta))\ (\text{old global error}) + \text{local error}_n \quad .$$

Therefore, the errors will be magnified if

$$|1 + h_n f_y(\eta)| > 1$$

and we say that the Euler method is unstable in this case.

Note: We use the word stable in two ways: stability of an ODE and stability region for a method for solving ODEs.

So the stability interval for Euler's method is

$$-2 < hf_y < 0$$

for a single equation. For a system of equations, the stability region is the region where the eigenvalues of $\mathbf{I} + h\mathbf{f}_y$ are in the unit circle.

Backward Euler method

How to derive it

Geometric derivation:

picture

Find y_{n+1} so that

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

Taylor series derivation

$$y(t) = y(t+h) - hy'(t+h) + \frac{h^2}{2}y''(\xi)$$

where $\xi \in [t, t+h]$, so

$$y_{n+1} = y_n + hf_{n+1}$$

How to use it

Given y_0 and $t_0, t_1, \ldots t_N$,

For $n = 0, \ldots, N-1$,

Solve the nonlinear equation

$$y_{n+1} = y_n + (t_{n+1} - t_n)f(t_{n+1}, y_{n+1})$$

Example: $y' = -y$

$$y_{n+1} = y_n - h_n y_{n+1}$$

so

$$(1 + h_n)y_{n+1} = y_n$$

and

$$y_{n+1} = y_n/(1 + h_n)\,.$$

In general, it is not this easy to solve the nonlinear equation, and we use a nonlinear equation solver from Chapter 8, or we use functional iteration:

P (predict): Guess y_{n+1} (perhaps using Euler's method).

E (evaluate): Evaluate $f_{n+1} = f(t_{n+1}, y_{n+1})$.

C (correct): Plug the current guess in, to get a new guess:

$$y_{n+1} = y_n + h_n f_{n+1}\,.$$

E: Evaluate $f_{n+1} = f(t_{n+1}, y_{n+1})$.

Repeat the CE steps if necessary.

We call this a PECE (or PE(CE)^k) scheme.

Note: If we fail to solve the nonlinear equation exactly, this adds an additional source of error.

Jargon

If y_{n+1} is on the right-hand side of the equation, we say that the ODE solver is implicit. Example: Backward Euler.

If y_{n+1} is not on the right-hand side of the equation, we say that the ODE solver is explicit. Example: Euler.

Question: Implicit methods certainly cause more work. Are they worth it?

Answer: Their stability properties allow us to solve problems that are not easy with explicit methods.

Local error

Taylor series says that the local error is

$$\frac{h^2}{2} y''(\xi)$$

This is first order, just like Euler's method.

Global error

We know

$$y(t_{n+1}) = y(t_n) + h_n f(t_{n+1} y(t_{n+1})) + \frac{h^2}{2} y''(\xi_n), \quad n = 0, 1, 2, \ldots,$$

and

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1})\,.$$

Therefore,

$$y(t_{n+1}) - y_{n+1} = y(t_n) - y_n + h_n(f(t_{n+1}, y(t_{n+1})) - f(t_{n+1}, y_{n+1})) + \frac{h^2}{2} y''(\xi_n)\,.$$

- The plum-colored terms are global errors.
- The black term is the local error.
- Using the MVT, the blue term can be written as

$$h_n(f(t_{n+1}, y(t_{n+1})) - f(t_{n+1}, y_{n+1})) = h_n f_y(\eta)(y(t_{n+1}) - y_{n+1})\,.$$

So we have the expression

$$(1 - h_n f_y(\eta))\ \text{(new global error)} = \text{(old global error)} + \text{local error}_n \quad .$$

so

$$\text{new global error} = (1 - h_n f_y(\eta))^{-1}[\ \text{(old global error)} + \text{local error}_n] \quad .$$

Therefore, the errors will be magnified if

$$|1 - h_n f_y(\eta)| < 1$$

and we say that the backward Euler method is unstable in this case.

So the stability interval for Backward Euler's method is

$$h f_y < 0 \ \text{ or } \ h f_y > 2$$

for a single equation. For a system of equations, the stability region is the region where all eigenvalues of $\mathbf{I} - h\mathbf{f}_y$ are outside the unit circle.

Example: Backward Euler is stable on the equation $y' = -y$ for all positive h.

Backward Euler is unstable for $y' = y$ when h is small and positive and inaccurate when h is large.

The Adams family

This family comes in two varieties:

- Adams-Bashforth methods (like Euler)
- Adams-Moulton methods (like backward Euler)

Adams-Bashforth Methods

Idea:

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

and we approximate the integrand by polynomial interpolation at the points

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

for some integer k.

Example: If $k = 1$, we interpolate the integrand by a polynomial of degree 0, with $p(t_n) = f_n$, so

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_n dt \\ &= y_n + (t_{n+1} - t_n) f_n \,, \end{aligned}$$

which is Euler's method. []

Example: If $k = 2$, we interpolate the integrand by a polynomial of degree 1, with

$$\begin{aligned} p(t_n) &= f_n \\ p(t_{n-1}) &= f_{n-1} \end{aligned}$$

so

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n-1} + \frac{f_n - f_{n-1}}{t_n - t_{n-1}}(t - t_{n-1})dt \\ &= y_n + h_n f_{n-1} + \frac{f_n - f_{n-1}}{h_{n-1}} \frac{(t_{n+1} - t_{n-1})^2 - (t_n - t_{n-1})^2}{2} \,. \end{aligned}$$

[]

For more formulas, the textbook.

Adams-Moulton Methods

Idea:

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

and we approximate the integrand by polynomial interpolation at the points

$$(t_{n+1}, f_{n+1}), \ldots, (t_{n-k+2}, f_{n-k+2})$$

for some integer k.

Example: If $k = 1$, we interpolate the integrand by a polynomial of degree 0, with $p(t_{n+1}) = f_{n+1}$, so

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n+1} dt \\ &= y_n + (t_{n+1} - t_n) f_{n+1} \,, \end{aligned}$$

which is backward Euler's method. []

Example: If $k = 2$, we interpolate the integrand by a polynomial of degree 1, with

$$\begin{aligned} p(t_n) &= f_n \\ p(t_{n+1}) &= f_{n+1} \end{aligned}$$

so

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n+1} + \frac{f_n - f_{n+1}}{t_n - t_{n+1}} (t - t_{n+1}) dt \\ &= y_n + \frac{h_n}{2} (f_{n+1} + f_n) \end{aligned}$$

which is a generalization of the Trapezoidal rule for integration.

Unquiz: Show the steps that get us from the first equation to the second. []

For more formulas, see the textbook.

How are the Adams formulas used?

We use one of the Adams-Bashforth formulas as a predictor, and the matching Adams-Moulton formula as a corrector, just as we did with the Euler and backward-Euler formulas.

Note that the two families share function evaluations, so the cost per step is only 2 evaluations of f for a PECE scheme, regardless of the order of the methods.

To make a PECE scheme practical, we need to add stepsize control and error estimation.

Some ingredients in building a practical ODE solver

The Adams family

This family comes in two varieties:

- Adams-Bashforth methods (like Euler)

- Adams-Moulton methods (like backward Euler)

Adams-Bashforth Methods

$$
\begin{aligned}
y_{n+1} &= y_n + hf_n \text{ error : } \frac{h^2}{2}y^{(2)}(\eta) \\
y_{n+1} &= y_n + \frac{h}{2}(3f_n - f_{n-1}) \text{ error : } + \frac{5h^3}{12}y^{(3)}(\eta) \\
y_{n+1} &= y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \text{ error : } \frac{3h^4}{8}y^{(4)}(\eta)
\end{aligned}
$$

For more formulas, see the textbook.

Adams-Moulton Methods

$$
\begin{aligned}
y_{n+1} &= y_n + hf_{n+1} - \frac{h^2}{2}y^{(2)}(\eta) \\
y_{n+1} &= y_n + \frac{h}{2}(f_n + f_{n+1}) - \frac{h^3}{12}y^{(3)}(\eta) \\
y_{n+1} &= y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}) - \frac{h^4}{24}y^{(4)}(\eta)
\end{aligned}
$$

For more formulas, see the textbook.

How are the Adams formulas used?

P (predict): Guess y_{n+1} using one of the Adams-Bashforth formulas.

E (evaluate): Evaluate $f_{n+1} = f(t_{n+1}, y_{n+1})$.

C (correct): Plug the current guess in, to get a new guess:

$$y_{n+1} = y_n + \alpha f_{n+1} + \dots .$$

using one of the Adams-Moulton formulas.

E: Evaluate $f_{n+1} = f(t_{n+1}, y_{n+1})$.

Error control

We have two tools to help us keep the size of the local error under control:

- We can change the stepsize h.
- We can change the order k.

Estimating the size of the error

Recall how we estimated error for numerical integration:

- We used 2 different methods to estimate the integral.
- We took the difference between the two estimates, and used that difference as our estimate of the error.

Programs for solving ODEs use a similar trick. We use 2 different methods to estimate y_{n+1}:

- perhaps a predictor and a corrector.
- or perhaps two methods of different order.

The difference between the estimates, or perhaps a weighted difference, is an estimate of the error.

So we can get an estimate of the error and compare it with the tolerance provided by the user.

Changing the stepsize

Suppose the error estimate is much too large: Then we can reduce the stepsize (usually by a factor of 2) and try again.

Suppose the error estimate is much smaller than needed: Then we can increase the stepsize (usually doubling it) and save ourselves some work when we take the next step.

Changing the order of the method

In general, we can take a bigger step using a higher order method, unless the solution is badly behaved.

When we take our first step with an Adams method, we need to use Euler's method, because we have no history.

When we take our second step, we can use the second-order Adams methods, because we have information about one old point.

As we keep walking, we can use higher order methods, by saving information about old function values.

But there is a limit to the usefulness of history; generally, fifth order methods are about as high as we go.

Some Unquizzes

- Write Matlab code to start an Adams algorithm. Begin using Euler's method, and gradually increase the order of the method to 3.
- Write Matlab code to check the error and decide whether to change h.
- Write Matlab code to decide whether to cut back to a lower order method.

Runge-Kutta methods

These methods build on the idea we used in the Euler method of predicting a new value by walking along a tangent to the curve. They just do it in a more complicated way.

Example of a Runge-Kutta method: Suppose we let

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \alpha h, y_n + \beta k_1) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

where we choose α, β, a, b to make the formula as accurate as possible.

We are going to expand everything in Taylor Series, and match as many terms as possible.

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

but

$$\begin{aligned} y'(t_n) &= f(t_n, y_n) \equiv f \\ y''(t_n) &= \frac{\partial f}{\partial t}(t_n, y_n) + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}(t_n, y_n) \equiv f_t + f_y f \end{aligned}$$

so

$$y(t_{n+1}) = y(t_n) + fh + (f_t + f_y f)\frac{h^2}{2} + O(h^3)\,.$$

Now we expand k_1 and k_2:

$$\begin{aligned} k_1 &= hf \\ k_2 &= h[f(t_n, y_n) + \alpha h f_t + \beta k_1 f_y + O(h^2)] \\ &= hf + \alpha h^2 f_t + \beta k_1 h f_y + O(h^3) \end{aligned}$$

so

$$\begin{aligned} y_{n+1} &= y_n + ak_1 + bk_2 \\ &= y_n + ahf + b(hf + \alpha h^2 f_t + \beta h f h f_y) + O(h^3) \\ &= y_n + (a+b)hf + \alpha b h^2 f_t + \beta b h^2 f_y f + O(h^3) \end{aligned}$$

We want to match the coefficients in this expansion to the coefficients in the (plum-colored) Taylor Series expansion of y:

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

so

$$\begin{aligned} a + b &= 1 \\ \alpha b &= \frac{1}{2} \\ \beta b &= \frac{1}{2} \end{aligned}$$

There are many solutions (in fact, an infinite number). One of them is (Heun's method)

$$\begin{aligned} a &= \frac{1}{2} \\ b &= \frac{1}{2} \\ \alpha &= 1 \\ \beta &= 1 \end{aligned}$$

This choice gives a second order Runge-Kutta method.

Note that the work per step is 2 f-evaluations.

The most useful Runge-Kutta method is one of order 4, given in the textbook. It requires 4 f-evaluations per step, and many pages for its derivation.

Final words

- Runge-Kutta methods require a lot of function evaluations per step.
- My preference is to use predictor-corrector methods from the Adams family for nonstiff systems.
- There is a Gear family, somewhat like the Adams family, for handling stiff systems. See the ODE solvers in Matlab whose name ends in the letter s for solvers for stiff systems.
- Stiff solvers also work on non-stiff systems, but they will be more expensive than Adams methods in this case.