Numerical Solutions for DEs HW2

YANG, Ze (5131209043)

March 22, 2017

A Note to TA:
Hi, this is the senior student from Antai College who did not register for this course. I would like to do all the assignments for practice, but feel free to just skip my homework if you don't have time. Thank you again for allowing me to access the assignments and other class material! :)
- Ze

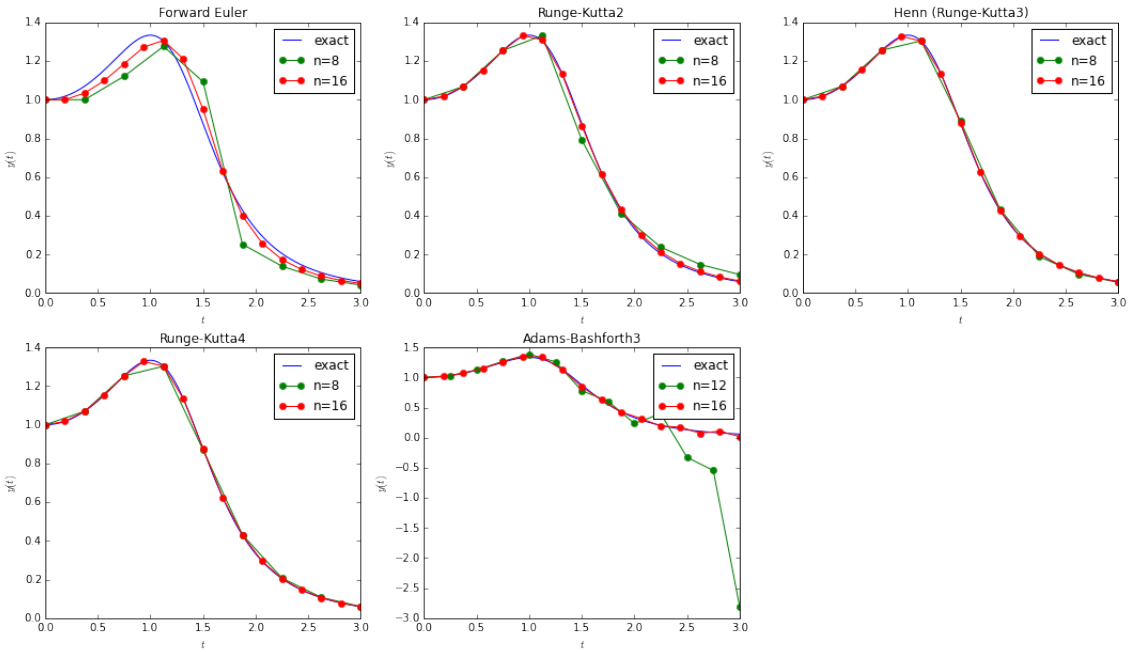
Problem 1. a. Implement *RK2*, *Heun's Method* and the classical *RK4*, and justify the rate of convergence numerically.

b. Choose appropriate Runge-Kutta method to initialize *Adams-Bashforth* method of order 3,

$$y_{n+3} = y_{n+2} + h[\tfrac{23}{12} f(t_{n+2}, y_{n+2}) - \tfrac{4}{3} f(t_{n+1}, y_{n+1}) + \tfrac{5}{12} f(t_n, y_n)]$$

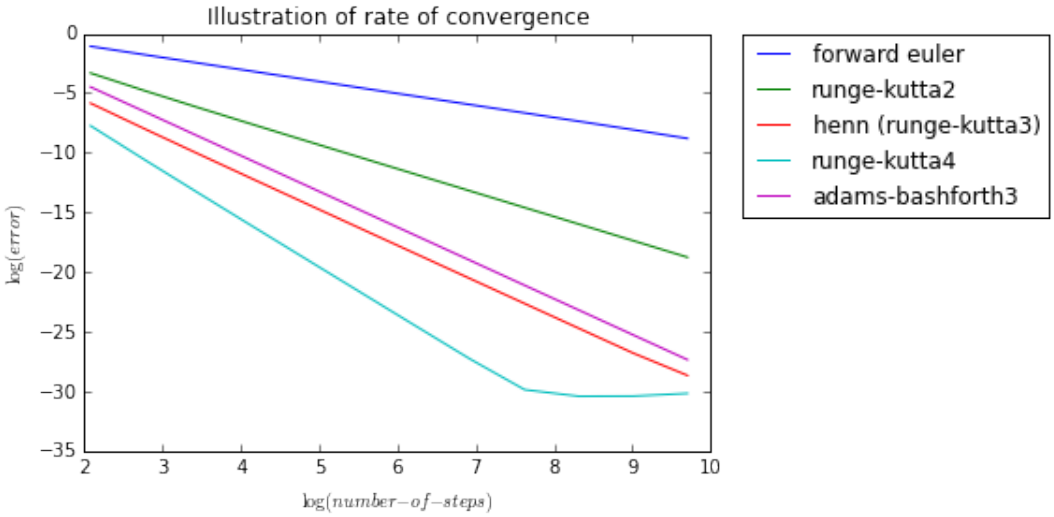
Justify the rate of convergence numerically.

Solution. We solve the test case 6: $y' = t^2 - y$ with all the five methods above.



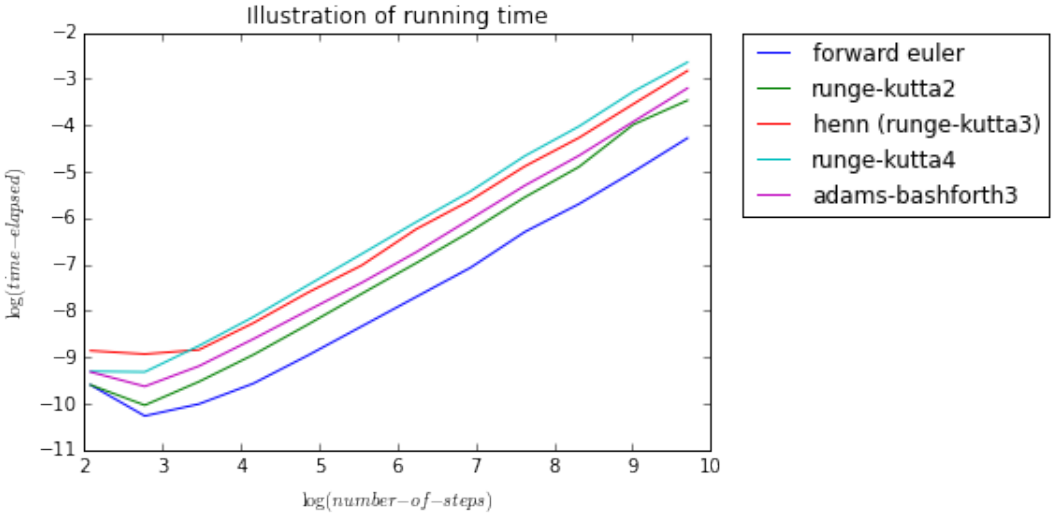
Where the *Adams-Bashforth* multistep method is intialized with a 3-order method (Heun's method).

We plot $\log(n)$ against $\log(error)$ to check the rate of convergence. Theoretically, for a method of order k, we expect to get a line with slope $-k$.



From the figure above we know that it is indeed the case. The curve for 4-th order Runge-Kutta method becomes flat in the end because the round-off error will dominate the error of the algorithm for n sufficiently large. And from the figure we know that the round-off error may be $\sim 2^{-30}$.

We also investigate the running time. The slope of these curves are 1, because all the algorithms take $O(n)$ time, regardless of their orders.



Please also see the attached code for details.

Problem 2. Consider the RK method (I just use the notations in the class)

$$\frac{y_{n+1}-y_n}{h} = c_1k_1 + c_2k_2$$

where $k_1 = f(t_n, y_n), k_2 = f(t_n + b_{21}h, y_n + b_{21}hk_1)$, and $c_1, c_2, b_{21} \in \mathbb{R}$.

a. Show that there is a choice of these parameters such that the truncation error of the method

$$\frac{T_n}{\Delta t} = \frac{y(t_{n+1}) - y(t_n)}{h} - c_1 f(t_n, y(t_n)) - c_2 f(t_n + b_{21}h, y(t_n) + b_{21}hf(t_n, y(t_n)))$$

is of order 2 as $h \to 0$.

b. Suppose that a second order method of the above form is applied to the IVP $y' = -\lambda y$, $y(0) = 1$, where λ is *real positive* number. Show that the sequence $\{y_n\}_{n\geq 0}$ is bounded $\iff$ $h \leq 2/\lambda$ (like the interval of absolute stability). Show further that, for such λ: $|y(t_n) - y_n| \leq \frac{1}{6}\lambda^3 h^2 t_n$, $n \geq 0$.

Proof. (a) We have (by f we mean $f(t_n, y(t_n))$, and by y we mean $y(t_n)$):

$$\begin{aligned} y' &= f \\ y'' &= f_t + f_y y' = f_t + f_y f \\ y''' &= f_{tt} + f_{ty}y' + y''f_y + (f_y)'y' = f_{tt} + 2f_{ty}f + f^2 f_{yy} + f_y f_t + f_y^2 f \end{aligned} \tag{1}$$

Therefore use taylor expansion of $y(t_{n+1})$ at t_n:

$$\frac{y(t_{n+1}) - y(t_n)}{h} = f + \frac{h}{2}(f_t + f_y f) + \frac{h^2}{6}(f_{tt} + 2f_{ty}f + f^2 f_{yy} + f_y f_t + f_y^2 f) \tag{2}$$

And that

$$\begin{aligned} b_1k_1 + b_2k_2 &= b_1 f + b_2 f(t_n + c_2 h, y(t_n) + h a_{21} f(t_n, y(t_n))) \\ &= b_1 f + b_2(f + f_t c_2 h + f_y h a_{21} f + \tfrac{1}{2}(f_{tt}c_2^2 h^2 + 2f_{ty}c_2 h \cdot a_{21} h f + f_{yy} h^2 a_{21}^2 f^2 + O(h^3)) \\ &= (b_1 + b_2) f + h(b_2 c_2 f_t + b_2 a_{21} f_y f) + \tfrac{h^2}{2}(c_2^2 f_{tt} + 2c_2 a_{21} f_{ty} f + a_{21} f_{yy} f^2) + O(h^3) \end{aligned} \tag{3}$$

So we subtract $b_1k_1 + b_2k_2$ from $\frac{y(t_{n+1})-y(t_n)}{h}$. To achieve a truncation error of order 2, we have to let $O(1)$ and $O(h)$ terms equal to 0:

$$\begin{aligned} &O(1) \text{ terms}: f - (b_1 + b_2) f = 0 \\ &O(h) \text{ terms}: \frac{h}{2}(f_t + f_y f) - h(b_2 c_2 f_t + b_2 a_{21} f_y f) = 0 \\ &\Rightarrow b_1 + b_2 = 1; \quad b_2 c_2 = \tfrac{1}{2}; \quad b_2 a_{21} = \tfrac{1}{2} \end{aligned} \tag{4}$$

Clearly, there exists such parameters. For example, $b_1 = b_2 = \frac{1}{2}$, $a_{21} = c_2 = 1$. □

(b.) Apply RK2 to $y' = -\lambda y$: $k_1 = -\lambda y_n$. $k_2 = -\lambda(y_n + a_{21} h k_1)$. So let $z := \lambda h$

$$y_{n+1} = y_n + h(-b_1\lambda y_n - b_2\lambda y_n + b_2 a_{21} h \lambda^2 y_n) = y_n(1 - b_1 z - b_2 z + b_2 a_{21} z^2) = y_n(1 - z + \tfrac{1}{2}z^2)$$

Therefore $y_n = (1 - z + \frac{1}{2}z^2)^n y_0$ is bounded $\Rightarrow |1 - z + \frac{1}{2}z^2| \leq 1$. The solution is $0 \leq z \leq 2$, i.e. $h \leq 2/\lambda$. The exact solution $y(t_n) = e^{-\lambda t} y(0) = e^{-nz}$. Hence

$$y(t_n) - y_n = e^{-nz} - (1 - z + \tfrac{1}{2}z^2)^n$$

Use taylor expansion to e^{-nz}, and trinomial expansion to the second term:

$$\begin{aligned} e^{-nz} &= 1 - nz + \frac{n^2 z^2}{2} - \frac{n^3 z^3}{6} + \frac{n^4 z^4}{24} + h.o.t. \\ (1 - z + \tfrac{1}{2}z^2)^n &= 1 + n(-z) + n(\tfrac{z^2}{2}) + \tfrac{n(n-1)}{2!}(-z)^2 + \tfrac{n(n-1)}{1!1!}(-z)(\tfrac{z^2}{2}) + \tfrac{n(n-1)(n-2)}{3!}(-z)^3 \\ &\quad + \tfrac{n(n-1)(n-2)(n-3)}{4!}(-z)^4 + \tfrac{n(n-1)}{2!}(\tfrac{z^2}{2})^2 + \tfrac{n(n-1)(n-2)}{2!1!}(-z)^2(\tfrac{z^2}{2}) + h.o.t. \\ &= 1 - nz + \tfrac{(n^2-n+n)}{2}z^2 - \tfrac{[(n^3-3n^2+2n)+(3n^2-3n)]}{6}z^3 \\ &\quad + \tfrac{[(n^4-6n^3+11n^2-6n)+(3n^2-3n)+(6n^3-18n^2+12n)]}{24}z^4 + h.o.t \\ &= 1 - nz + \tfrac{n^2}{2}z^2 - \tfrac{n^3-n}{6}z^3 + \tfrac{n^4-4n^2+3n}{24}z^4 + h.o.t. \end{aligned} \tag{5}$$

Hence

$$\begin{aligned} y(t_n) - y_n &= -\frac{n}{6}z^3 + \frac{4n^2 - 3n}{24}z^4 + h.o.t. \\ &= -\frac{1}{6}\lambda^3 h^2 t_n + \frac{4n^2 - 3n}{24}z^4 + h.o.t. \end{aligned} \tag{6}$$

For $n \in \mathbb{N}^+ \cup \{0\}$, we have $4n^2 - 3n \geq 0$. Therefore

$$|y(t_n) - y_n| = \left| -\frac{1}{6}\lambda^3 h^2 t_n + \frac{4n^2 - 3n}{24}z^4 + h.o.t. \right| \leq \left| \frac{1}{6}\lambda^3 h^2 t_n \right|$$

As $h \to 0$, $z = \lambda h \to 0$.

□

Problem 3. Find the general solution $\{y_n\}$ for the homogeneous recurrence reltation

$$a_0 y_n + a_1 y_{n+1} + ... + a_s y_{n+s} = 0$$

Proof. WLOG we assume the recurrence is of order s, i.e. $a_s \neq 0$. Then rearrange the terms as:

$$y_{n+s} + \tfrac{a_{s-1}}{a_s} y_{n+s-1} + ... + \tfrac{a_1}{a_s} y_{n+1} + \tfrac{a_0}{a_s} y_{n+1} = y_{n+s} + c_{s-1} y_{n+s-1} + ... + c_1 y_{n+1} + c_0 y_n = 0$$

with $c_k = \frac{a_k}{a_s}$, $k = 0, 1, ..., s-1$. We start with a lemma on *companion matrix*.

Lemma. The chracteristic polynomial of n-degree $P(\lambda) = \lambda^s + c_{s-1}\lambda^{s-1} + ... + c_1\lambda + c_0 = \det(\lambda \boldsymbol{I} - \boldsymbol{C})$, where

$$\boldsymbol{C} = \begin{pmatrix} -c_{s-1} & -c_{s-2} & \dots & -c_1 & -c_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

Proof of Lemma: We prove by induction. For degree $s = 1$ case, $P(\lambda) = \lambda + c_0$. And $\lambda \boldsymbol{I} - \boldsymbol{C}$ is just $\lambda - (-c_0)$.
Now assume the relation is true for $s-1$, at s, we have $\det(\lambda \boldsymbol{I} - \boldsymbol{C}) =$

$$\begin{vmatrix} c_{s-1} & c_{s-2} & \dots & c_1 & c_0 \\ -1 & \lambda & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & \lambda \end{vmatrix}_s = \lambda \begin{vmatrix} c_{s-1} & c_{s-2} & \dots & c_2 & c_1 \\ -1 & \lambda & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & \lambda \end{vmatrix}_{s-1} + (-1)^{s+1} c_0 \begin{vmatrix} -1 & \lambda & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & \lambda \\ 0 & 0 & \dots & 0 & -1 \end{vmatrix}_{s-1} \quad (\dagger)$$

with our inductive assumption, the determinant in the first term is exactly $\det(\lambda \boldsymbol{I} - \boldsymbol{C})$ in the degree $s-1$ case. So,

$$(\dagger) = \lambda(\lambda^{s-1} + c_{s-1}\lambda^{s-2} + ... + c_2\lambda + c_1) + (-1)^{s+1} c_0 (-1)^{s-1} = P(\lambda) \quad \blacksquare$$

The lemma suggests that finding the roots of $P(\lambda) = 0$ is equivalent to finding the eigenvalues of $\boldsymbol{C}$.
We denote $\boldsymbol{y}_n = (y_{n+s}, ..., y_{n+2}, y_{n+1})^\top$; $\boldsymbol{c} = (c_{s-1}, ..., c_1, c_0)^\top$, and unit vector with 1 on position k as $\boldsymbol{\iota}_k$. We find that the linear recurrence can be vectorized as $y_{n+s} = -\boldsymbol{c}^\top \boldsymbol{y}_{n-1}$. Moreover $y_{n+k} = \boldsymbol{\iota}_k^\top \boldsymbol{y}_{n-1}$ for $k = 1, ..., s-1$.

$$\boldsymbol{y}_n = \begin{pmatrix} y_{n+s} \\ y_{n+s-1} \\ y_{n+s-2} \\ \vdots \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{c}^\top \boldsymbol{y}_n \\ \boldsymbol{\iota}_1^\top \boldsymbol{y}_n \\ \boldsymbol{\iota}_2^\top \boldsymbol{y}_n \\ \vdots \\ \boldsymbol{\iota}_{s-1}^\top \boldsymbol{y}_n \end{pmatrix} = \begin{pmatrix} -c_{s-1} & -c_{s-2} & \dots & -c_1 & -c_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} y_{n+s-1} \\ y_{n+s-2} \\ y_{n+s-3} \\ \vdots \\ y_n \end{pmatrix} = \boldsymbol{C}\boldsymbol{y}_{n-1} = \boldsymbol{C}^n \boldsymbol{y}_0$$

where $\boldsymbol{y}_0 = (y_s, y_{s-1}, ..., y_2, y_1)^\top$ is the initial condition.
Assume $\boldsymbol{C}$ has eigenvalues $\lambda_1, ..., \lambda_p$, $p \leq s$ with algebraic multiplicity $m_1, ..., m_p$, where $\sum_p m_p = s$. Hence we decompose $\boldsymbol{C} = \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{-1}$ in its Jordan Normal Form:

$$\boldsymbol{C}^n \boldsymbol{y}_0 = \boldsymbol{P}\boldsymbol{J}^n\boldsymbol{P}^{-1}\boldsymbol{y}_0 = \boldsymbol{P} \begin{pmatrix} \boldsymbol{J}_{\lambda_1}^n & & & \\ & \boldsymbol{J}_{\lambda_2}^n & & \\ & & \ddots & \\ & & & \boldsymbol{J}_{\lambda_p}^n \end{pmatrix} \boldsymbol{P}^{-1}\boldsymbol{y}_0 \quad (*)$$

$$\text{where } \boldsymbol{J}_{\lambda_i}^n = \begin{pmatrix} \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} & \binom{n}{2}\lambda_i^{n-2} & \dots & \binom{n}{m_i-1}\lambda_i^{n-(m_i-1)} \\ & \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} & \dots & \binom{n}{m_i-2}\lambda_i^{n-(m_i-2)} \\ & & \ddots & \ddots & \vdots \\ & & & \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} \\ & & & & \lambda_i^n \end{pmatrix}_{m_i \times m_i}$$

The typical entry is $\binom{n}{r}\lambda_i^{n-r} = \frac{\binom{n}{r}}{\lambda_i^r}\lambda_i^n$, which we can regard as $p_i^{[r]}(n)\lambda_i^n$. Here $p_i^{[r]}(n)$ is a polynomial of n to the degree r. Therefore, we regard $(*)$ as applying linear opeartions on these entries. We know that applying linear operations on a collection of polynomials $\{p_i^{[r]}(n)\}_{r=1}^{m_i-1}$ will result in another polynomial with degree no more than $m_i - 1$, denote it as $g_i^{[m_i-1]}(n)$. So we have

$$\begin{aligned} y_{n+s} &= \sum_{i=1}^{p} g_i^{[m_i-1]}(n)\lambda_i^n = \sum_{i=1}^{p}\sum_{r=0}^{m_i-1} \alpha_{ir} n^r \lambda_i^n \\ &= (\alpha_{10} + \alpha_{11}n + ... + \alpha_{1,m_1-1}n^{m_1-1})\lambda_1^n + ... + (\alpha_{p0} + \alpha_{p1}n + ... + \alpha_{p,m_p-1}n^{m_p-1})\lambda_p^n \end{aligned} \tag{7}$$

Where $\{\alpha_{ir}\}_{i=1,r=0}^{p,m_i-1}$ are constants. Since $\sum_{i=1}^{p} m_i = s$, there are s constants in total, which are solved by inserting initial condition $\boldsymbol{y}_0 = (y_s, y_{s-1}, ..., y_1)^\top$ into the general solution. If we index the initial condition as $(y_0, y_{-1}, ..., y_{-(s-1)})^\top$ instead, the general solution will be

$$y_n = (C_{10} + C_{11}n + ... + C_{1,m_1-1}n^{m_1-1})\lambda_1^n + ... + (C_{p0} + C_{p1}n + ... + C_{p,m_p-1}n^{m_p-1})\lambda_p^n$$

Where $\{C_{ir}\}$ is another set of constants. By lemma, $\{\lambda_i\}$ are the roots of $P(\lambda) = 0$, with multiplicity $\{m_i\}$. And this general solution holds for all $n \geq 0$.

□

Problem 4. Find the region of A-stablity for the multistep method:

$$y_{n+2} - y_n = \frac{1}{3}h[f(t_{n+2}, y_{n+2}) + 4f(t_{n+1}, y_{n+1}) + f(t_n, y_n)]$$

Proof. The first- and second-characteristic polynomial of the method is:

$$\rho(z) = z^2 - z; \quad \sigma(z) = \tfrac{1}{3}z^2 + \tfrac{4}{3}z + \tfrac{1}{3}$$

So the method is absolute stable $\iff$ the roots z of $\Pi(z, \bar{h}) = \rho(z) - \bar{h}\sigma(z)$ have modulus < 1.

$$\Pi(z, \bar{h}) = z^2 - z - \bar{h}\left(\tfrac{1}{3}z^2 + \tfrac{4}{3}z + \tfrac{1}{3}\right) = (1 - \tfrac{\bar{h}}{3})z^2 + (-1 - \tfrac{4\bar{h}}{3})z + (-\tfrac{\bar{h}}{3}) = 0 \tag{8}$$

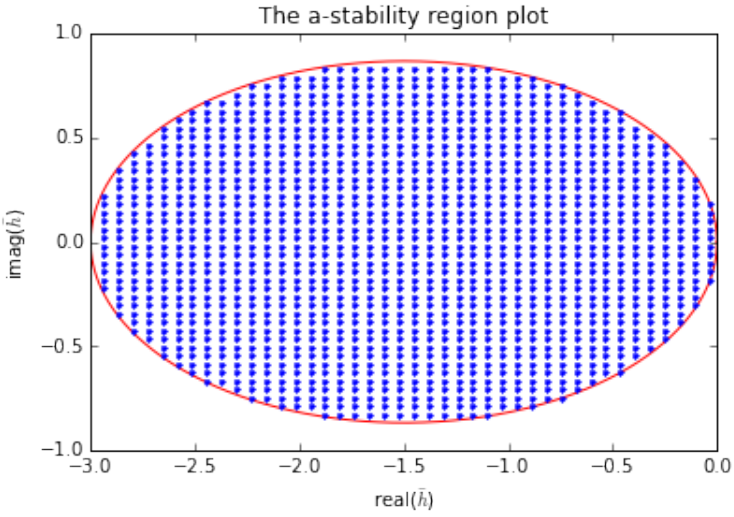
We solve the equation and obtain

$$z = \frac{\pm\sqrt{3(4\bar{h}^2 + 12\bar{h} + 3)} - 4\bar{h} - 3}{2(\bar{h} - 3)}$$

Or $\bar{h} = (z^2 - z)/(\frac{1}{3}z^2 + \frac{4}{3}z + \frac{1}{3})$. Denote the stability region as $\mathcal{S}$, then on $\partial\mathcal{S}$, the polynomial has root $|z| = 1$. Hence we can find the boundary by

$$\partial\mathcal{S} = \left\{\frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})}, \theta \in [0, 2\pi]\right\} = \left\{\frac{e^{2i\theta} - e^{i\theta}}{\frac{1}{3}e^{2i\theta} + \frac{4}{3}e^{i\theta} + \frac{1}{3}}, \theta \in [0, 2\pi]\right\}$$

The region is give as the following plot (please also see the attached code).



□

Problem 5. (Iserles 3.4) Apply the classical RK4 method (with coefficient table shown as below) to scalar autonomous ode $y' = f(y)$, check that this method is indeed of order 4.

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ c_4 & a_{41} & a_{42} & a_{43} & \\ \hline & b_1 & b_2 & b_3 & b_4 \end{array} = \begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Proof. We want to calculate the truncation error:

$$\frac{T_n}{h} = \frac{y(t_{n+1}) - y(t_n)}{h} - (b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4)$$

We first calculate the first term using taylor expansion. We calculate the derivatives with `Mathematica`:

$$\begin{aligned} y' = f(y(t)) = f, \quad \tfrac{1}{2!}y^{(2)} &= y'(t)f'(y(t)) = \tfrac{1}{2}f_yf \\ \tfrac{1}{3!}y^{(3)} &= \tfrac{1}{6}y'(t)^2f''(y(t)) + \tfrac{1}{6}y''(t)f'(y(t)) = \tfrac{1}{6}f_{yy}f^2 + \tfrac{1}{6}f_y^2f \\ \tfrac{1}{4!}y^{(4)} &= \tfrac{1}{24}[f^{(3)}(y(t))y'(t)^3 + 3y'(t)y''(t)f''(y(t)) + y^{(3)}(t)f'(y(t))] \\ &= \tfrac{1}{24}[f_{yyy}f^3 + 3f(ff_y)f_{yy} + (f^2f_{yy} + f_y^2f)f_y] \\ &= \tfrac{1}{24}f_{yyy}f^3 + \tfrac{1}{6}f_{yy}f_yf^2 + \tfrac{1}{24}f_y^3f \end{aligned} \tag{9}$$

Then we expand the second part, the k's are given by

$$\begin{aligned} k_1 &= f \\ k_2 &= f(y_n + \tfrac{1}{2}hf(y_n)) = \{f + f_y\cdot(\tfrac{1}{2}hf) + \tfrac{f_{yy}}{2}\cdot(\tfrac{1}{2}hf))^2 + \tfrac{f_{yyy}}{6}\cdot(\tfrac{1}{2}hf)^3 + O(h^4)\} \\ &= f + \tfrac{1}{2}f_yfh + \tfrac{1}{8}f_{yy}f^2h^2 + \tfrac{1}{48}f_{yyy}f^3h^3 + O(h^4) \end{aligned} \tag{10}$$

$$\begin{aligned} k_3 &= f\left(y_n + \tfrac{1}{2}h\left(f + \tfrac{1}{2}f_yfh + \tfrac{1}{8}f_{yy}f^2h^2 + \tfrac{1}{48}f_{yyy}f^3h^3 + O(h^4)\right)\right) \\ &= \Big\{f + f_y\cdot\tfrac{1}{2}h\left(f + \tfrac{1}{2}f_yfh + \tfrac{1}{8}f_{yy}f^2h^2 + O(h^3)\right) + \tfrac{f_{yy}}{2}\cdot\tfrac{1}{4}h^2\left(f + \tfrac{1}{2}f_yfh + O(h^2)\right)^2 + \\ &\quad + \tfrac{f_{yyy}}{6}\cdot\tfrac{1}{8}h^3\left(f + O(h)\right)^3 + O(h^4)\Big\} \\ &= f + h\left(\tfrac{f_yf}{2} + \tfrac{1}{4}f_y^2fh + \tfrac{1}{16}f_{yy}f_yf^2h^2 + O(h^3)\right) + h^2\left(\tfrac{f_{yy}f^2}{8} + \tfrac{1}{8}f_{yy}f_yf^2h + O(h^2)\right) \\ &\quad + h^3\left(\tfrac{1}{48}f_{yyy}f^3 + O(h)\right) + O(h^4) \\ &= f + \tfrac{1}{2}f_yfh + (\tfrac{1}{4}f_y^2f + \tfrac{1}{8}f_{yy}f^2)h^2 + (\tfrac{3}{16}f_{yy}f_yf^2 + \tfrac{1}{48}f_{yyy}f^3)h^3 + O(h^4) \end{aligned} \tag{11}$$

$$\begin{aligned} k_4 &= f\left(y_n + h\left[f + \tfrac{1}{2}f_yfh + (\tfrac{1}{4}f_y^2f + \tfrac{1}{8}f_{yy}f^2)h^2 + (\tfrac{3}{16}f_{yy}f_yf^2 + \tfrac{1}{48}f_{yyy}f^3)h^3 + O(h^4)\right]\right) \\ &= \Big\{f + f_y\cdot h\left[f + \tfrac{1}{2}f_yfh + (\tfrac{1}{4}f_y^2f + \tfrac{1}{8}f_{yy}f^2)h^2 + O(h^3)\right] + \tfrac{f_{yy}}{2}\cdot h^2\left[f + \tfrac{1}{2}f_yfh + O(h^2)\right]^2 \\ &\quad + \tfrac{f_{yyy}}{6}\cdot h^3\left[f + O(h)\right]^3 + O(h^4)\Big\} \\ &= f + h\left[f_yf + \tfrac{1}{2}f_y^2fh + (\tfrac{1}{4}f_y^3f + \tfrac{1}{8}f_{yy}f_yf^2)h^2 + O(h^3)\right] + h^2\left[\tfrac{1}{2}f_{yy}f^2 + \tfrac{1}{2}f_{yy}f_yf^2h + O(h^2)\right] \\ &\quad + h^3(\tfrac{1}{6}f_{yyy}f^3 + O(h)) + O(h^4) \\ &= f + f_yfh + (\tfrac{1}{2}f_y^2f + \tfrac{1}{2}f_{yy}f^2)h^2 + (\tfrac{1}{4}f_y^3f + \tfrac{5}{8}f_{yy}f_yf^2 + \tfrac{1}{6}f_{yyy}f^3)h^3 + O(h^4) \end{aligned} \tag{12}$$

Hence we have:

$$\begin{aligned} \tfrac{1}{6}k_1 &= \tfrac{1}{6}f \\ \tfrac{1}{3}k_2 &= \tfrac{1}{3}f + \tfrac{1}{6}f_yfh + \tfrac{1}{24}f_{yy}f^2h^2 + \tfrac{1}{144}f_{yyy}f^3h^3 + O(h^4) \\ \tfrac{1}{3}k_3 &= \tfrac{1}{3}f + \tfrac{1}{6}f_yfh + (\tfrac{1}{12}f_y^2f + \tfrac{1}{24}f_{yy}f^2)h^2 + (\tfrac{1}{16}f_{yy}f_yf^2 + \tfrac{1}{144}f_{yyy}f^3)h^3 + O(h^4) \\ \tfrac{1}{6}k_4 &= \tfrac{1}{6}f + \tfrac{1}{6}f_yfh + (\tfrac{1}{12}f_y^2f + \tfrac{1}{12}f_{yy}f^2)h^2 + (\tfrac{1}{24}f_y^3f + \tfrac{5}{48}f_{yy}f_yf^2 + \tfrac{1}{36}f_{yyy}f^3)h^3 + O(h^4) \end{aligned} \tag{13}$$

$$\begin{aligned}\tfrac{1}{6}k_1+\tfrac{1}{3}k_2+\tfrac{1}{3}k_3+\tfrac{1}{6}k_4 &= (\tfrac{1}{6}+\tfrac{1}{3}+\tfrac{1}{3}+\tfrac{1}{6})f+(\tfrac{1}{6}+\tfrac{1}{6}+\tfrac{1}{6})f_yfh\\ &\quad+[(\tfrac{1}{24}+\tfrac{1}{24}+\tfrac{1}{12})f_{yy}f^2+(\tfrac{1}{12}+\tfrac{1}{12})f_y^2f]h^2\\ &\quad+[(\tfrac{1}{144}+\tfrac{1}{144}+\tfrac{1}{36})f_{yyy}f^3+(\tfrac{1}{16}+\tfrac{5}{48})f_{yy}f_yf^2+(\tfrac{1}{24})f_y^3f]h^3+O(h^4)\\ &= f+\tfrac{1}{2}f_yfh+(\tfrac{1}{6}f_{yy}f^2+\tfrac{1}{6}f_y^2f)h^2+(\tfrac{1}{24}f_{yyy}f^3+\tfrac{1}{6}f_{yy}f_yf^2+\tfrac{1}{24}f_y^3f)h^3+O(h^4)\quad(\dagger)\end{aligned} \tag{14}$$

Compare (†) with the Taylor expansion of $\frac{y(t_{n+1})-y(t_n))}{h}$, and notice that the coefficients of $O(1), O(h), O(h^2)$ and $O(h^3)$ match together, hence they vanish. The truncation error $T_n/h = O(h^4)$. □

Problem 6. (Iserles 3.5) Suppose that a ν-stage explicit RK method of order ν is applied to linear scalar ode $y' = \lambda y$, show that

$$y_n = \left(\sum_{k=0}^{\nu}\frac{1}{k!}(h\lambda)^k\right)^n y_0, \qquad n = 0, 1, ...$$

Proof. The scheme of explicit RK method is

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h)$$

And the truncation error is

$$\frac{T_n}{h} = \frac{y(t_{n+1})-y(t_n)}{h} - \Phi(t_n, y(t_n); h)$$

As usual we expand the first term via taylor series, and since y is solution of the linear scalar ode, we have $y'' = \lambda y' = \lambda^2 y$, ..., $y^{(n)} = \lambda^n y$.

$$\begin{aligned}\frac{y(t_{n+1})-y(t_n)}{h} &= y'(t_n)+\frac{y''(t_n)}{2!}h+...+\frac{y^{(\nu)}(t_n)}{\nu!}h^{\nu-1}+O(h^\nu)\\ &= y(t_n)\left(\lambda+\frac{\lambda^2}{2!}h+...+\frac{\lambda^\nu}{\nu!}h^{\nu-1}\right)+O(h^\nu)\\ &= \frac{y(t_n)}{h}\sum_{k=1}^{\nu}\frac{1}{k!}(\lambda h)^k+O(h^\nu)\end{aligned} \tag{15}$$

Since the method is of order ν, the truncation error is $O(h^\nu)$, which implies that $\Phi(t_n, y(t_n); h) = \frac{y(t_n)}{h}\sum_{k=1}^{\nu}\frac{1}{k!}(\lambda h)^k$. Therefore the RK method formula is

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h) = y_n + y_n\sum_{k=1}^{\nu}\frac{1}{k!}(\lambda h)^k = y_n\sum_{k=0}^{\nu}\frac{1}{k!}(\lambda h)^k$$

i.e.

$$y_n = \left(\sum_{k=1}^{\nu}\frac{1}{k!}(\lambda h)^k\right)^n y_0$$

□

Problem 7. (Iserles 4.1) Let $\boldsymbol{y}' = \boldsymbol{\Lambda y}$, $\boldsymbol{y}(t_0) = \boldsymbol{y}_0$ be solved (with a constant step size $h > 0$) by a one-step method with a function $r(\cdot)$ that obeys the relation (4.12). Suppose that a nonsingular matrix $\boldsymbol{V}$ and a diagonal matrix $\boldsymbol{D}$ exists such that $\boldsymbol{\Lambda} = \boldsymbol{VDV}^{-1}$, show that there exists vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_d \in \mathbb{R}^d$ such that

$$\boldsymbol{y}(t_n) = \sum_{j=1}^{d} e^{t_n\lambda_j}\boldsymbol{x}_j, \qquad n = 0, 1, ...$$

And

$$\boldsymbol{y}_n = \sum_{j=1}^{d}[r(h\lambda_j)]^n\boldsymbol{x}_j, \qquad n = 0, 1, ...$$

where $\lambda_1, ..., \lambda_d$ are eigenvalues of $\boldsymbol{\Lambda}$. Deduce that the values of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, given in (4.3) and (4.4) are identical.

Proof. (a.) The exact solution of the linear ode system is $\boldsymbol{y}(t) = e^{\boldsymbol{\Lambda} t}\boldsymbol{y}_0$. Since $\boldsymbol{\Lambda}$ is diagonalizable, we have $\boldsymbol{\Lambda} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}$, where the entries of $\boldsymbol{D}$ are the d linear independent eigenvalues of $\boldsymbol{\Lambda}$ and the columns of $\boldsymbol{V}$ are the corresponding eigenvectors. Therefore

$$e^{\boldsymbol{\Lambda} t} = e^{\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}t} = \sum_{k=0}^{\infty}\frac{t^k}{k!}(\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1})^k = \sum_{k=0}^{\infty}\frac{t^k}{k!}\boldsymbol{V}\boldsymbol{D}^k\boldsymbol{V}^{-1} = \boldsymbol{V}\sum_{k=0}^{\infty}\frac{t^k}{k!}\boldsymbol{D}^k\boldsymbol{V}^{-1} = \boldsymbol{V}e^{\boldsymbol{D}t}\boldsymbol{V}^{-1}$$

We denote $\boldsymbol{w} = \boldsymbol{V}^{-1}\boldsymbol{y}_0 = (w_1, ..., w_d)^\top$ is a $d \times 1$ vector, then

$$\boldsymbol{y}(t_n) = \boldsymbol{V}e^{\boldsymbol{D}t}\boldsymbol{V}^{-1}\boldsymbol{y}_0 = \boldsymbol{V}\begin{pmatrix} e^{\lambda_1 t_n} & & & \\ & e^{\lambda_2 t_n} & & \\ & & \ddots & \\ & & & e^{\lambda_d t_n}\end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d\end{pmatrix} = \boldsymbol{V}\begin{pmatrix} w_1 e^{\lambda_1 t_n} \\ w_2 e^{\lambda_2 t_n} \\ \vdots \\ w_d e^{\lambda_d t_n}\end{pmatrix} = \sum_{j=1}^{d} w_j e^{\lambda_j t_n}\boldsymbol{v}_j$$

where $\boldsymbol{w} = \boldsymbol{V}^{-1}\boldsymbol{y}_0$ and $\boldsymbol{v}_j$ is the j-th eigenvector. If we denote $\boldsymbol{x}_j := w_j\boldsymbol{v}_j$, then we come to the desired results. $\boldsymbol{y}(t_n) = \sum_{j=1}^{d} e^{\lambda_j t_n}\boldsymbol{x}_j$. □

(b.) We apply the ν-stage RK method on this linear ode system, the formulation of IRK is given by (c.f. Iserles eq.3.9)

$$\begin{cases}\boldsymbol{\xi}_j = \boldsymbol{y}_n + h\sum_{i=1}^{\nu} a_{ji}\boldsymbol{f}(t_n + c_i h, \boldsymbol{\xi}_i) \quad j = 1, 2, ..., \nu \\ \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{j=1}^{\nu} b_j \boldsymbol{f}(t_n + c_j h, \boldsymbol{\xi}_j)\end{cases} \tag{16}$$

And apply $\boldsymbol{f}(\cdot, \boldsymbol{y}) = \boldsymbol{\Lambda}\boldsymbol{y}$, the scheme becomes

$$\begin{cases}\boldsymbol{\xi}_j = \boldsymbol{y}_n + h\boldsymbol{\Lambda}\sum_{i=1}^{\nu} a_{ji}\boldsymbol{\xi}_i \quad j = 1, 2, ..., \nu \\ \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{\Lambda}\sum_{j=1}^{\nu} b_j \boldsymbol{\xi}_j\end{cases} \tag{17}$$

Notice that:

- $\sum_{j=1}^{\nu} b_j\boldsymbol{\xi}_j$ is a linear combination of $\{\boldsymbol{\xi}_j\}_1^\nu$s, with weights being $\boldsymbol{b} = (b_1, ..., b_\nu)^\top$. So we write it as $\boldsymbol{\Xi}\boldsymbol{b}$, where $\boldsymbol{\Xi}$ is $d \times \nu$ matrix, whose columns are the vectors $\{\boldsymbol{\xi}_j\}_1^\nu$s. Then the RK iteration formula can be written as
$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{b} \quad (\dagger)$$
- $\sum_{i=1}^{\nu} a_{ji}\boldsymbol{\xi}_i$ can be considered in the same way, i.e. $\sum_{i=1}^{\nu} a_{ji}\boldsymbol{\xi}_i = \boldsymbol{\Xi}\boldsymbol{a}_j$, $\boldsymbol{a}_j = (a_{j1}, a_{j2}, ..., a_{j\nu})^\top$. Stack $\boldsymbol{\xi}_j = \boldsymbol{y}_n + h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{a}_j$ together on columns:
$$\boldsymbol{\Xi} = \big(\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_\nu\big) = \big(\boldsymbol{y}_n + h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{a}_1, ..., \boldsymbol{y}_n + h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{a}_\nu\big) = \boldsymbol{y}_n\boldsymbol{1}^\top + h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{A} \tag{18}$$
Where $\boldsymbol{1}^\top = (1, 1, .., 1)$ is an $1 \times \nu$ iota vector. So we have $\boldsymbol{y}_n\boldsymbol{1}^\top = \boldsymbol{\Xi} - h\boldsymbol{\Lambda}\boldsymbol{\Xi}\boldsymbol{A} = (\boldsymbol{I_d} - h\boldsymbol{\Lambda})\boldsymbol{\Xi}(\boldsymbol{I}_\nu - \boldsymbol{A})$, where $\boldsymbol{I}_d, \boldsymbol{I}_\nu$ are $d \times d$ and $\nu \times \nu$ identity matrices respectively. Assume $\boldsymbol{I}_d - h\boldsymbol{\Lambda}$ and $\boldsymbol{I}_\nu - \boldsymbol{A}$ are nonsingular, we have
$$\boldsymbol{\Xi} = (\boldsymbol{I}_d - h\boldsymbol{\Lambda})^{-1}\boldsymbol{y}_n\boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1} \quad (\ddagger)$$

Insert (‡) into (†), using diagonalization $\boldsymbol{\Lambda} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}$:

$$\begin{aligned}\boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + h\boldsymbol{\Lambda}(\boldsymbol{I}_d - h\boldsymbol{\Lambda})^{-1}\boldsymbol{y}_n\boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1}\boldsymbol{b} \\ &= \boldsymbol{y}_n + h\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}\left(\boldsymbol{V}(\boldsymbol{I}_d - h\boldsymbol{D})\boldsymbol{V}^{-1}\right)^{-1}\boldsymbol{y}_n\boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1}\boldsymbol{b} \\ &= \boldsymbol{V}\boldsymbol{I}_d\boldsymbol{V}^{-1}\boldsymbol{y}_n \cdot 1 + h\boldsymbol{V}\boldsymbol{D}(\boldsymbol{I}_d - h\boldsymbol{D})^{-1}\boldsymbol{V}^{-1}\boldsymbol{y}_n\boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1}\boldsymbol{b} \\ &= \boldsymbol{V}(\boldsymbol{I}_d + h\boldsymbol{D}(\boldsymbol{I}_d - h\boldsymbol{D})^{-1})\boldsymbol{V}^{-1}\boldsymbol{y}_n(1 + \boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1}\boldsymbol{b})\end{aligned} \tag{19}$$

Notice that:

- $(1 + \boldsymbol{1}^\top(\boldsymbol{I}_\nu - \boldsymbol{A})^{-1}\boldsymbol{b})$ is just a scalar, totally determined by the coefficients of the RK method. We denote it as ζ and bring it forward, i.e. consider $\boldsymbol{y}_{n+1} = \boldsymbol{V}\zeta(\boldsymbol{I}_d + h\boldsymbol{D}(\boldsymbol{I}_d - h\boldsymbol{D})^{-1})\boldsymbol{V}^{-1}\boldsymbol{y}_n$
- $\zeta(\boldsymbol{I}_d + h\boldsymbol{D}(\boldsymbol{I}_d - h\boldsymbol{D})^{-1})$ is a diagonal matrix. And denote $\boldsymbol{D}_{\bar{h}} = h\boldsymbol{D} = \mathrm{diag}\{h\lambda_1, ..., h\lambda_d\}$, this matrix is $\zeta(\boldsymbol{I}_d + \boldsymbol{D}_{\bar{h}}(\boldsymbol{I}_d - \boldsymbol{D}_{\bar{h}})^{-1})$. By the lemma on the book, it must be of the form:
$$\zeta(\boldsymbol{I}_d + \boldsymbol{D}_{\bar{h}}(\boldsymbol{I}_d - \boldsymbol{D}_{\bar{h}})^{-1}) = \begin{pmatrix} r(h\lambda_1) & & \\ & \ddots & \\ & & r(h\lambda_d)\end{pmatrix} \tag{20}$$
$r(\cdot)$ is the function of the type defined in (Iserles) section 4.3.

Therefore, by the discussion above, and using the same notation in our derivation of the exact solution:

$$\begin{aligned}\boldsymbol{y}_n &= \boldsymbol{V}\begin{pmatrix} r(h\lambda_1) & & \\ & \ddots & \\ & & r(h\lambda_d)\end{pmatrix}\boldsymbol{V}^{-1}\boldsymbol{y}_{n-1} = \boldsymbol{V}\begin{pmatrix} r(h\lambda_1) & & \\ & \ddots & \\ & & r(h\lambda_d)\end{pmatrix}^n\boldsymbol{V}^{-1}\boldsymbol{y}_0 \\ &= \boldsymbol{V}\begin{pmatrix} r^n(h\lambda_1) & & \\ & \ddots & \\ & & r^n(h\lambda_d)\end{pmatrix}\begin{pmatrix} w_1 \\ \vdots \\ w_d\end{pmatrix} = \boldsymbol{V}\begin{pmatrix} w_1[r(h\lambda_1)]^n \\ \vdots \\ w_d[r(h\lambda_d)]^n\end{pmatrix} = \sum_{j=1}^{d}[r(h\lambda_j)]^n w_j\boldsymbol{v}_j\end{aligned} \tag{21}$$

Which equals to $\sum_{j=1}^{d}[r(h\lambda_j)]^n\boldsymbol{x}_j$ with same $\boldsymbol{x}_j = w_j\boldsymbol{v}_j$, finished the proof.

□

Problem 8. (Iserles 4.2) Consider the solution of $\boldsymbol{y}' = \boldsymbol{\Lambda}\boldsymbol{y}$ where

$$\boldsymbol{\Lambda} = \begin{pmatrix}\lambda & 1 \\ 0 & \lambda\end{pmatrix}, \quad \lambda \in \mathbb{C}^-$$

a. Show that
$$\boldsymbol{\Lambda}^n = \begin{pmatrix}\lambda^n & n\lambda^{n-1} \\ 0 & \lambda^n\end{pmatrix}, \quad n = 0, 1, ...$$

b. Let g be an arbitrary function that is analytic about the origin. The 2×2 matrix $g(\boldsymbol{\Lambda})$ can be defined by substituting powers of $\boldsymbol{\Lambda}$ into the Taylor expansion of g, show that
$$g(t\boldsymbol{\Lambda}) = \begin{pmatrix}g(t\lambda) & tg'(t\lambda) \\ 0 & g(t\lambda)\end{pmatrix}$$

c. By letting $g(z) = e^z$, show that $\lim_{t\to\infty}\boldsymbol{y}(t) = \boldsymbol{0}$.

d. Suppose that $\boldsymbol{y}' = \boldsymbol{\Lambda}\boldsymbol{y}$ is solved with RK method, using a constant step size $h > 0$. Let r be the function from lemma 4.1. Letting $g = r$, obtain the explicit form of $[r(h\boldsymbol{\Lambda})]^n$, $n = 0, 1, ...$

e. Show that if $h\lambda \in \mathcal{D}$, where $\mathcal{D}$ is the linaer stability domain of the RK method, then $\lim_{n\to\infty}\boldsymbol{y}_n = \boldsymbol{0}$.

Proof. (a.) We can decompose $\boldsymbol{\Lambda} = \lambda\boldsymbol{I} + \boldsymbol{B}$, with $\boldsymbol{B} = \begin{pmatrix}0 & 1 \\ 0 & 0\end{pmatrix}$. It is easy to see that $\boldsymbol{B}^n = 0$ for $n \geq 2$. Hence

$$\boldsymbol{\Lambda}^n = (\lambda\boldsymbol{I})^n + \binom{n}{1}(\lambda\boldsymbol{I})^{n-1}\boldsymbol{B} = \begin{pmatrix}\lambda^n & n\lambda^{n-1} \\ 0 & \lambda^n\end{pmatrix} \quad \square$$

(b.) By the given definition of $g(t\boldsymbol{\Lambda})$ (with $\boldsymbol{\Lambda}^0 := \boldsymbol{I}$, $\boldsymbol{\Lambda}^n$ being the expression we obtained in (a)):

$$\begin{aligned}g(t\boldsymbol{\Lambda}) &= \sum_{n=0}^{\infty}\frac{g^{(n)}(0)}{n!}(t\boldsymbol{\Lambda})^n = \sum_{n=0}^{\infty}\frac{g^{(n)}(0)}{n!}\begin{pmatrix}(t\lambda)^n & nt(t\lambda)^{n-1} \\ 0 & (t\lambda)^n\end{pmatrix} \\ &= \begin{pmatrix}\sum_{n=0}^{\infty}\frac{g^{(n)}(0)}{n!}(t\lambda)^n & t\sum_{n=0}^{\infty}\frac{g^{(n)}(0)}{(n-1)!}(t\lambda)^{n-1} \\ 0 & \sum_{n=0}^{\infty}\frac{g^{(n)}(0)}{n!}(t\lambda)^n\end{pmatrix} \\ &= \begin{pmatrix}g(t\lambda) & tg'(t\lambda) \\ 0 & g(t\lambda)\end{pmatrix} \quad \square\end{aligned} \tag{22}$$

(c.) The solution of ode is $\boldsymbol{y}(t) = e^{\boldsymbol{\Lambda}t}\boldsymbol{y}_0$. Let $g(\boldsymbol{\Lambda}t) = e^{\boldsymbol{\Lambda}t}$, by the result of (b.) we get:

$$\boldsymbol{y}(t) = e^{\boldsymbol{\Lambda}t}\boldsymbol{y}_0 = \begin{pmatrix}e^{\lambda t} & te^{\lambda t} \\ 0 & e^{\lambda t}\end{pmatrix}\boldsymbol{y}_0 \tag{23}$$

Since $\lambda \in \mathbb{C}^-$, it is clear that $\lim_{t\to\infty}e^{\lambda t} = 0$, and is of higher order of decay than the order of growth of t, i.e. $\lim_{t\to\infty}te^{\lambda t} = 0$ too. Hence in the expression above, $\lim_{t\to\infty}e^{\boldsymbol{\Lambda}t} = \boldsymbol{O}$, so $\lim_{t\to\infty}e^{\Lambda t}\boldsymbol{y}_0 = \boldsymbol{0}$.

(d.) By the result of b., we have

$$[r(h\mathbf{\Lambda})]^n = [g(h\mathbf{\Lambda})]^n = \begin{pmatrix} g(h\lambda) & hg'(h\lambda) \\ 0 & g(h\lambda) \end{pmatrix}^n \qquad (*) \tag{24}$$

And note that this is of the same form as $\mathbf{\Lambda}$ itself, we decompose

$$(*) = \left[g(h\lambda)\boldsymbol{I} + \begin{pmatrix} 0 & hg'(h\lambda) \\ 0 & 0 \end{pmatrix} \right]^n = \begin{pmatrix} [g(h\lambda)]^n & n[g(h\lambda)]^{n-1} hg'(h\lambda) \\ 0 & [g(h\lambda)]^n \end{pmatrix} = \begin{pmatrix} [r(h\lambda)]^n & n[r(h\lambda)]^{n-1} hr'(h\lambda) \\ 0 & [r(h\lambda)]^n \end{pmatrix} \quad \square \tag{25}$$

(e.) If $h\lambda \in \mathcal{D}$ (the stability region of RK method), by lemma 4.2 of (Iserles), $\mathcal{D} = \{z : |r(z)| < 1\}$. So we have $|r(h\lambda)| < 1$. Hence

$$\lim_{n\to\infty} [r(h\lambda)]^n = 0 \quad \text{and} \quad \lim_{n\to\infty} n[r(h\lambda)]^{n-1} = 0$$

So, $\lim_{n\to\infty} \boldsymbol{y}_n = \lim_{n\to\infty} [r(h\mathbf{\Lambda})]^n \boldsymbol{y}_0 = \mathbf{0}$, finished the proof. $\square$

HW2 Code

March 22, 2017

1 Numerical Solutions for DEs HW2

YANG, Ze (5131209043)

Note to TA: Hi, this is the senior student from Antai College who did not register for this course. I would like to do all the assignments for practice, but feel free to just skip my homework if you don't have time.

Thank you again for allowing me to access the assignments and other class material! :)

Ze

1.1 Problem 1.

- Implement RK2

$$\begin{cases} y_{n+1} = y_n + hk_2 \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \end{cases} \tag{1}$$

- Heun Method

$$\begin{cases} y_{n+1} = y_n + h(\frac{1}{4}k_1 + \frac{3}{4}k_3) \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{1}{3}h, y_n + \frac{1}{3}hk_1) \\ k_3 = f(t_n + \frac{2}{3}h, y_n + \frac{2}{3}hk_1) \end{cases} \tag{2}$$

- The Classical RK4 Method

$$\begin{cases} y_{n+1} = y_n + h(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4) \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ k_3 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\ k_4 = f(t_n + h, y_n + hk_3) \end{cases} \tag{3}$$

- Choose appropriate RK method to initialize Adams-Bashforth method of order 3:

$$y_{n+3} = y_{n+2} + h(\frac{23}{12}f(t_{n+2}, y_{n+2}) - \frac{4}{3}f(t_{n+1}, y_{n+1}) + \frac{5}{12}f(t_n, y_n)) \tag{4}$$

And justify the rate of convergence.

```
In [1]: %matplotlib inline
        from __future__ import division
        import time
        import numpy as np
        import scipy.optimize
        import matplotlib.pyplot as plt
```

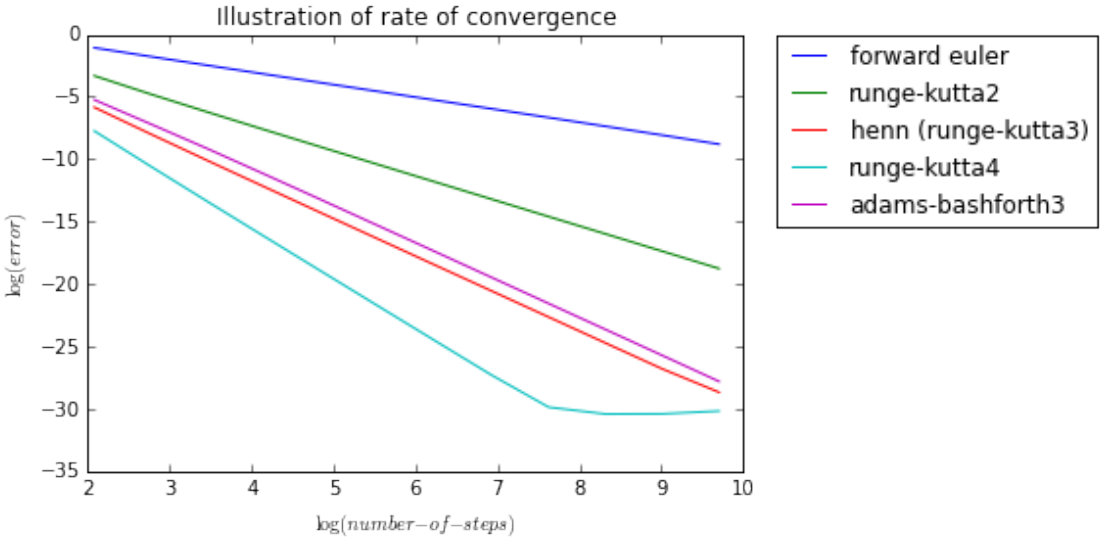
In [2]:
```python
# test cases
test_cases = {
    1: lambda t,y: np.pi * np.cos(np.pi * t),
    2: lambda t,y: t + y,
    3: lambda t,y: np.exp(t),
    4: lambda t,y: -20*y + 20* t + 1,
    5: lambda t,y: (y**2) * (t - t**3),
    6: lambda t,y: t**2 - y
}

# initial conditions
t0, t1, y0 = 0.0, 3.0, 1.0


# estimate error and performance
def err_est(method, f, n_sample, t0, t1, y0):
    """
    estimate the error of an numerical solution relative
    to an exact solution, which is obtained by using a very small h.

    @param method: function, the numerical method being used.
    @param f: a function of t and y, which is the derivative of y.
    @param n_sample: # of different choices of h that are tested.
    """
    nks, errs, t_elapsed = [], [], []
    # calculate "exact" solution using a very small h
    n_large = 2**18
    t_ex, y_ex = method(f, n_large, t0, t1, y0)

    for k in range(n_sample):
        n_k = 2**(k+3)
        s_k = int(n_large/n_k)
        start_time = time.time()
        # do the numerical procudure with given choice of h,n
        t_path, path = method(f, n_k, t0, t1, y0)
        t_elapsed.append(time.time() - start_time)
        # calc errors
        path_matched = [y_ex[s_k*i] for i in range(len(path))]
        err = max(np.abs(path - path_matched))
        errs.append(err)
        nks.append(n_k)
    return nks, errs, t_elapsed


# the benchmark, the method we implemented in last HW.
def explicit_euler_solve(f, n, t0, t1, y0):
    """
    explicit euler method, solve IVP y'(t)=f(t,y(t)), y(t0)=y0 by
    y_{n+1} <- y_{n} + hf(t_n, y_{n}).
    @param f: a function of t and y, which is the derivative of y.
    @param n: the number of steps.
    @param t0, y0: the initial value.
    @param t1: the other end to which we generate numerical solution
```

```
    @return t: the np.array {t_k}_1^n
    @return y: the np.array {y_k}_1^n
    """
    h = (t1 - t0) / n
    t, y = np.linspace(t0, t1, n+1), np.zeros(n+1)
    y[0] = y0
    for k in range(n):
        y[k+1] = y[k] + h*f(t[k], y[k])
    return t, y


def rk2(f, n, t0, t1, y0):
    """
    2-nd order RK method, solve IVP y'(t)=f(t,y(t)), y(t0)=y0 by
    k1 <- f(t_n, y_n)
    k2 <- f(t_n + h/2, y_n + hk_1/2)
    y_{n+1} <- y_n + hk_2.

    @param f: a function of t and y, which is the derivative of y.
    @param n: the number of steps.
    @param h: the step size.
    @param y0: the initial value.

    @return t: the np.array {t_k}_1^n
    @return y: the np.array {y_k}_1^n
    """
    h = (t1 - t0) / n
    t, y = np.linspace(t0, t1, n+1), np.zeros(n+1)
    y[0] = y0
    for k in range(n):
        k1 = f(t[k], y[k])
        k2 = f(t[k] + h/2, y[k] + h*k1/2)
        y[k+1] = y[k] + h*k2
    return t, y


def heun_rk3(f, n, t0, t1, y0):
    """
    3-rd order RK method, solve IVP y'(t)=f(t,y(t)), y(t0)=y0 by
    k1 <- f(t_n, y_n)
    k2 <- f(t_n + h/3, y_n + hk_1/3)
    k3 <- f(t_n + 2h/3, y_n + 2hk_2/3)
    y_{n+1} <- y_n + h(k_1/4 + 3k_3/4).

    @param f: a function of t and y, which is the derivative of y.
    @param n: the number of steps.
    @param h: the step size.
    @param y0: the initial value.

    @return t: the np.array {t_k}_1^n
    @return y: the np.array {y_k}_1^n
    """
    h = (t1 - t0) / n
    t, y = np.linspace(t0, t1, n+1), np.zeros(n+1)
```

```
    y[0] = y0
    for k in range(n):
        k1 = f(t[k], y[k])
        k2 = f(t[k] + h/3, y[k] + h*k1/3)
        k3 = f(t[k] + 2*h/3, y[k] + 2*h*k2/3)
        y[k+1] = y[k] + h*(k1/4 + 3*k3/4)
    return t, y


def rk4(f, n, t0, t1, y0):
    """
    the classical 4-th order RK method, solve IVP y'(t)=f(t,y(t)), y(t0)=y0 by
    k1 <- f(t_n, y_n)
    k2 <- f(t_n + h/2, y_n + hk_1/2)
    k3 <- f(t_n + h/2, y_n + hk_2/2)
    k4 <- f(t_n + h, y_n + hk_3)
    y_{n+1} <- y_n + h(k_1/6 + k_2/3 + k_3/3 + k_4/6).

    @param f: a function of t and y, which is the derivative of y.
    @param n: the number of steps.
    @param h: the step size.
    @param y0: the initial value.

    @return t: the np.array {t_k}_1^n
    @return y: the np.array {y_k}_1^n
    """
    h = (t1 - t0) / n
    t, y = np.linspace(t0, t1, n+1), np.zeros(n+1)
    y[0] = y0
    for k in range(n):
        k1 = f(t[k], y[k])
        k2 = f(t[k] + h/2, y[k] + h*k1/2)
        k3 = f(t[k] + h/2, y[k] + h*k2/2)
        k4 = f(t[k] + h, y[k] + h*k3)
        y[k+1] = y[k] + h*(k1/6 + k2/3 + k3/3 + k4/6)
    return t, y


def abf3(f, n, t0, t1, y0):
    """
    the 3-rd Adams Bashforth multistep method,
    solve IVP y'(t)=f(t,y(t)), y(t0)=y0 by

    y_{n+3} <- y_n + h(
        23/12 * f(t_{n+2}, y_{n+2})
        - 4/3 * f(t_{n+1}, y_{n+1})
        + 5/12 * f(t_n, y_n)
    ).

    the method uses a (higher than) 3-rd ordered one-step (rk3) method
    to generate another two initial values.
    Here we use the heun_rk3

    @param f: a function of t and y, which is the derivative of y.
```

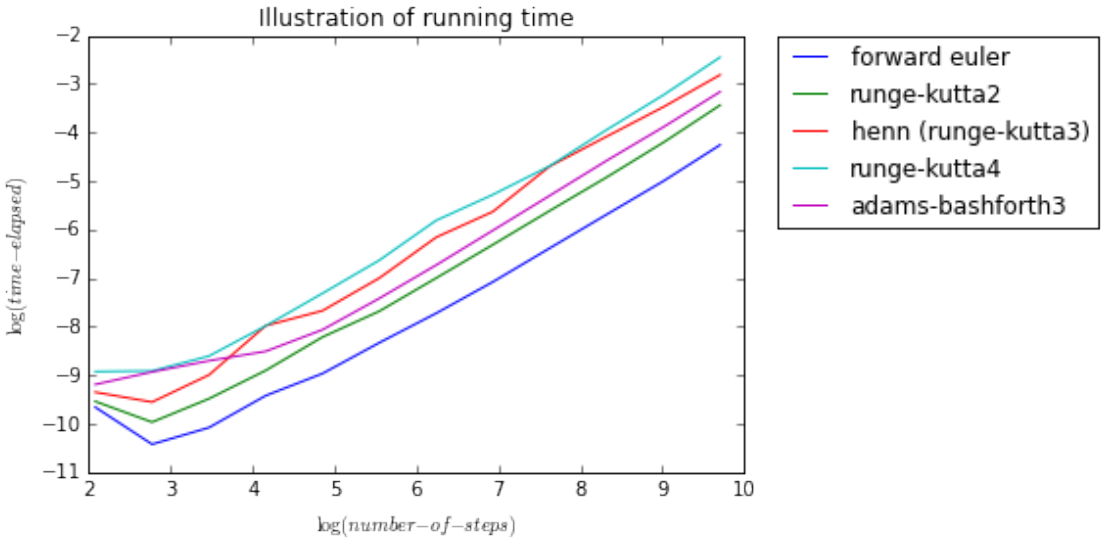
```
        @param n: the number of steps.
        @param h: the step size.
        @param y0: the initial value.

        @return t: the np.array {t_k}_1^n
        @return y: the np.array {y_k}_1^n
        """
        h = (t1 - t0) / n
        t_init, y_init = heun_rk3(f, 2, t0, t0+2*h, y0)
        t, y = np.linspace(t0, t1, n+1), np.zeros(n+1)
        y[0], y[1], y[2] = y0, y_init[1], y_init[2]
        for k in range(n-2):
            y[k+3] = y[k+2] + h*((23/12) * f(t[k+2], y[k+2]) +
                                 (-4/3) * f(t[k+1], y[k+1]) +
                                 (5/12) * f(t[k], y[k]))
        return t, y
```

In [3]:
```
if __name__ == '__main__':
    nks ,errs, t_elapsed = err_est(explicit_euler_solve, test_cases[6], 12, t0, t1, y0)
    nks2 ,errs2, t_elapsed2 = err_est(rk2, test_cases[6], 12, t0, t1, y0)
    nks3 ,errs3, t_elapsed3 = err_est(heun_rk3, test_cases[6], 12, t0, t1, y0)
    nks4 ,errs4, t_elapsed4 = err_est(rk4, test_cases[6], 12, t0, t1, y0)
    nks5 ,errs5, t_elapsed5 = err_est(abf3, test_cases[6], 12, t0, t1, y0)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(np.log(nks) ,np.log(errs))
    ax.plot(np.log(nks2) ,np.log(errs2))
    ax.plot(np.log(nks3) ,np.log(errs3))
    ax.plot(np.log(nks4) ,np.log(errs4))
    ax.plot(np.log(nks5) ,np.log(errs5))
    ax.set_title('Illustration of rate of convergence')
    ax.set_xlabel('$\log(number-of-steps)$')
    ax.set_ylabel('$\log(error)$')
    ax.legend(['forward euler', 'runge-kutta2', 'heun (runge-kutta3)', 'runge-kutta4', 'adams-ba
              bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```



In [4]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(np.log(nks), np.log(t_elapsed))
ax.plot(np.log(nks2), np.log(t_elapsed2))
ax.plot(np.log(nks3), np.log(t_elapsed3))
ax.plot(np.log(nks4), np.log(t_elapsed4))
ax.plot(np.log(nks5), np.log(t_elapsed5))
ax.set_title('Illustration of running time')
ax.set_xlabel('$\log(number-of-steps)$')
ax.set_ylabel('$\log(time-elapsed)$')
ax.legend(['forward euler', 'runge-kutta2', 'heun (runge-kutta3)', 'runge-kutta4', 'adams-ba
          bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

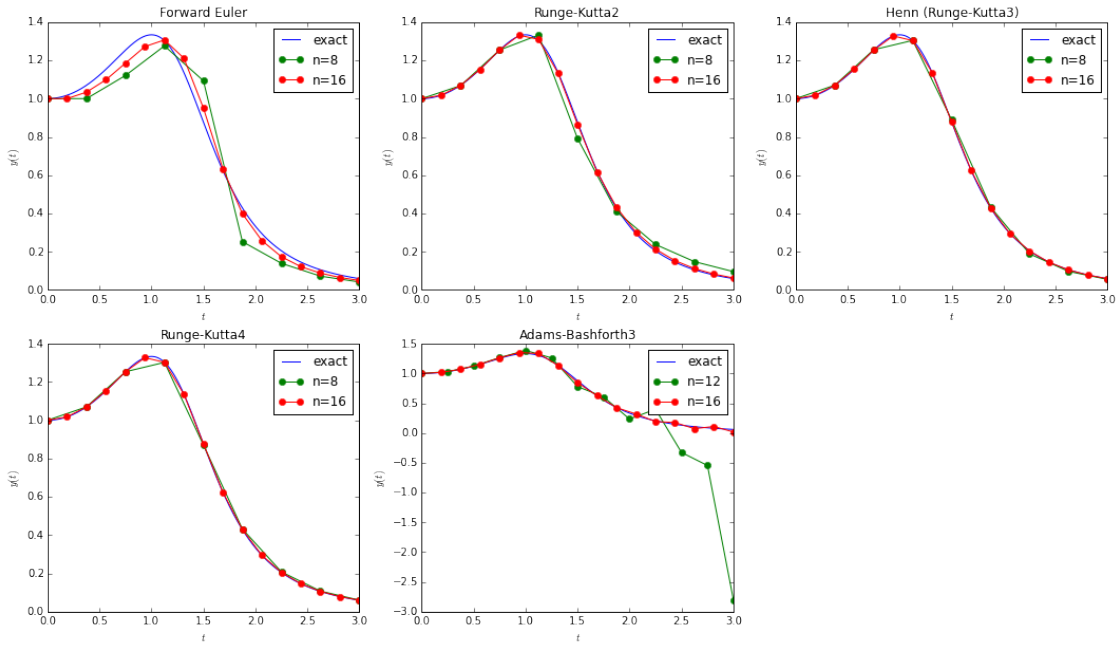
Out[4]: <matplotlib.legend.Legend at 0x107afc710>



In [5]:
```
# initial conditions
fig = plt.figure(figsize=(18,10))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(234)
ax5 = fig.add_subplot(235)

t_ex, y_ex = explicit_euler_solve(test_cases[5], 2**16, 0.0, 3.0, 1.00)
t1, y1 = explicit_euler_solve(test_cases[5], 8, 0.0, 3.0, 1.0)
t2, y2 = explicit_euler_solve(test_cases[5], 16, 0.0, 3.0, 1.0)
ax1.plot(t_ex, y_ex)
ax1.plot(t1, y1, '.-', markersize=12)
ax1.plot(t2, y2, '.-', markersize=12)
ax1.set_title('Forward Euler')
```

```
ax1.set_xlabel('$t$')
ax1.set_ylabel('$y(t)$')
ax1.legend(['exact', 'n=8', 'n=16'])

t_ex, y_ex = rk2(test_cases[5], 2**16, 0.0, 3.0, 1.00)
t1, y1 = rk2(test_cases[5], 8, 0.0, 3.0, 1.0)
t2, y2 = rk2(test_cases[5], 16, 0.0, 3.0, 1.0)
ax2.plot(t_ex, y_ex)
ax2.plot(t1, y1, '.-', markersize=12)
ax2.plot(t2, y2, '.-', markersize=12)
ax2.set_title('Runge-Kutta2')
ax2.set_xlabel('$t$')
ax2.set_ylabel('$y(t)$')
ax2.legend(['exact', 'n=8', 'n=16'])

t_ex, y_ex = heun_rk3(test_cases[5], 2**16, 0.0, 3.0, 1.00)
t1, y1 = heun_rk3(test_cases[5], 8, 0.0, 3.0, 1.0)
t2, y2 = heun_rk3(test_cases[5], 16, 0.0, 3.0, 1.0)
ax3.plot(t_ex, y_ex)
ax3.plot(t1, y1, '.-', markersize=12)
ax3.plot(t2, y2, '.-', markersize=12)
ax3.set_title('Heun (Runge-Kutta3)')
ax3.set_xlabel('$t$')
ax3.set_ylabel('$y(t)$')
ax3.legend(['exact', 'n=8', 'n=16'])

t_ex, y_ex = heun_rk3(test_cases[5], 2**16, 0.0, 3.0, 1.00)
t1, y1 = rk4(test_cases[5], 8, 0.0, 3.0, 1.0)
t2, y2 = rk4(test_cases[5], 16, 0.0, 3.0, 1.0)
ax4.plot(t_ex, y_ex)
ax4.plot(t1, y1, '.-', markersize=12)
ax4.plot(t2, y2, '.-', markersize=12)
ax4.set_title('Runge-Kutta4')
ax4.set_xlabel('$t$')
ax4.set_ylabel('$y(t)$')
ax4.legend(['exact', 'n=8', 'n=16'])

t_ex, y_ex = abf3(test_cases[5], 2**16, 0.0, 3.0, 1.00)
t1, y1 = abf3(test_cases[5], 12, 0.0, 3.0, 1.0)
t2, y2 = abf3(test_cases[5], 16, 0.0, 3.0, 1.0)
ax5.plot(t_ex, y_ex)
ax5.plot(t1, y1, '.-', markersize=12)
ax5.plot(t2, y2, '.-', markersize=12)
ax5.set_title('Adams-Bashforth3')
ax5.set_xlabel('$t$')
ax5.set_ylabel('$y(t)$')
ax5.legend(['exact', 'n=12', 'n=16'])
```

```
Out[5]: <matplotlib.legend.Legend at 0x108c26050>
```



1.2 Problem 4.

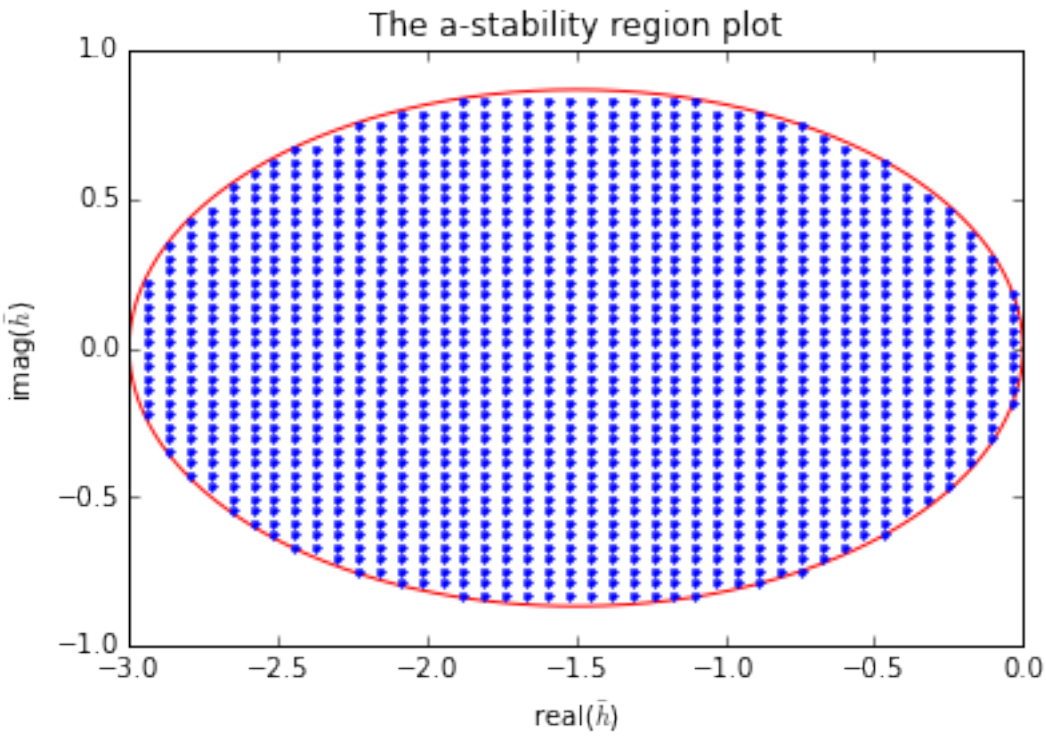
- Find the region of absolute stability for the multistep method

$$y_{n+2} - y_n = \tfrac{1}{3}h[f(t_{n+2}, y_{n+2}) + 4f(t_{n+1}, y_{n+1}) + f(t_n, y_n)]$$

```
In [6]: h1 = lambda z: (z**2 - z)/(z**2/3 +4*z/3 + 1/3)
```

```
def aStabRegion_boundary(h_bar_func):
    """
    Compute the boundary of absolute stability h_bar in the
    complex plane. Using the fact that
    \partial S = {h_bar(z) : |z| = 1}. We take z = exp(i*theta),
    with theta ranging from 0 to 2pi.

    @param h_bar_func: the functional of h_bar of z -> complex, z is the
    root to \Pi(z, h_bar) = 0.
    """
    theta = np.linspace(0, 2*np.pi, 1000)
    z = np.exp(theta*1j)
    h_bar = h_bar_func(z)
    return h_bar.real, h_bar.imag

z1 = lambda h: [
    (np.sqrt(12*h**2 + 36*h + 9) - 4*h - 3) / (2*h - 6),
    (-np.sqrt(12*h**2 + 36*h + 9) - 4*h - 3) / (2*h - 6),
]

def aStabRegion_mesh(x_lim, y_lim, root_funcs):
```

```
    """
    Collect the points on the mesh grids where the method
    is absolute stable. By testing whether all roots of \Pi(z, h_bar) = 0.
    have modulus smaller than 1.

    @param x_lim: tuple of 2 doubles, the real range on which the
    stability test is conducted.
    @param y_lim: tuple of 2 doubles, the imag range on which the
    stability test is conducted.
    @param root_funcs: list of functionals of h_bar -> complex,
    the roots of \Pi(z, h_bar) = 0.
    """
    x_in, y_in = [], []
    x_grids = np.linspace(x_lim[0], x_lim[1], 100)
    y_grids = np.linspace(y_lim[0], y_lim[1], 100)
    for x in x_grids:
        for y in y_grids:
            h = x + y*1j
            z_all = root_funcs(h)
            # conduct the test
            test = sum([np.abs(z)>=1 for z in z_all])
            if test == 0:
                x_in.append(x)
                y_in.append(y)
    return x_in, y_in

if __name__ == '__main__':
    x, y = aStabRegion_boundary(h1)
    x_pp, y_pp = aStabRegion_mesh([-4, 3], [-2, 2], z1)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(x, y, '-', color='r')
    ax.plot(x_pp, y_pp, '.')
    ax.set_title('The a-stability region plot')
    ax.set_xlabel('real($\\bar{h}$)')
    ax.set_ylabel('imag($\\bar{h}$)')
```


The a-stability region plot
1.0
0.5
0.0
−0.5
−1.0
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
imag($\bar{h}$)
real($\bar{h}$)

In [ ]: