Caleb Logemann
MATH 561 Numerical Analysis I
Homework 4

#1 (a) Determine the principle error function of the general explicit two-stage Runge-Kutta method.

The general explicit two-stage Runge-Kutta method can be described as follows.

$$\begin{aligned} k_1 &= f(x, y) \\ k_2 &= f(x + \mu h, y + \mu h k_1) \\ \Phi(x, y; h) &= \alpha_1 k_1 + \alpha_2 k_2 \end{aligned}$$

To find the priniciple error function, first the local truncation error must be found. The local truncation error is defined as

$$T(x, y; h) = \Phi(x, y; h) - \frac{1}{h}(y(x + h) - y(x))$$

The principle error function is the functional coefficient of h^p in the local truncation error, when p is the order of the method. Two-stage Runge-Kutta methods have in general an order of $p = 2$, so the principle error function is the coefficient of h^2. In order to find this the Taylor expansion of $\Phi(x, y; h)$ and $\frac{1}{h}(y(x + h) - y(x)))$ must be found, at least to the h^2 term.

First I will find the Taylor expansion of $\Phi(x, y; h) = \alpha_1 k_1 + \alpha_2 k_2$. The Taylor expansion of $k_1 = f(x, y)$ is just $f(x, y)$. The Taylor expansion of k_2 can be found as follows.

$$\begin{aligned} k_2 &= f(x + \mu h, y + \mu h k_1) \\ &= f(x + \mu h, y + \mu h f(x, y)) \\ &= f(x, y) + f_x(x, y)(\mu h) + f_y(x, y)(\mu h f(x, y)) \\ &+ \frac{1}{2}\Big(f_{xx}(x, y)(\mu h)^2 + 2 f_{xy}(x, y)(\mu^2 h^2 f(x, y)) + f_{yy}(x, y)(\mu^2 h^2 f(x, y)^2\Big) + O(h^3) \\ &= f(x, y) + \mu(f_x(x, y) + f(x, y) f_y(x, y))h \\ &+ \frac{1}{2}\mu^2\Big(f_{xx}(x, y) + 2 f(x, y) f_{xy}(x, y) + f(x, y)^2 f_{yy}(x, y)\Big)h^2 + O(h^3) \end{aligned}$$

Now the Taylor expansion of $\Phi(x, y; h)$ can be expressed as follows. Note that moving forward all values or derivatives of f will be evaluated at (x, y). Thus $f = f(x, y)$, $f_x = f_x(x, y)$, $f_y = f_y(x, y)$, and so on.

$$\begin{aligned} \Phi(x, y; h) &= \alpha_1 k_1 + \alpha_2 k_2 \\ &= \alpha_1 f + \alpha_2 \Big(f + \mu(f_x + f f_y)h + \frac{1}{2}\mu^2\Big(f_{xx} + f f_{xy} + f^2 f_{yy}\Big)h^2 + O(h^3)\Big) \\ &= (\alpha_1 + \alpha_2) f + \mu \alpha_2 (f_x + f f_y) h + \frac{1}{2}\alpha_2 \mu^2 \Big(f_{xx} + 2 f f_{xy} + f^2 f_{yy}\Big)h^2 + O(h^3) \end{aligned}$$

Now that the Taylor expansion of $\Phi(x, y; h)$ has been found the Taylor expansion of $\frac{1}{h}(y(x + h) - y(x)))$ must be found and put in terms of f.

$$\begin{aligned} y(x + h) &= y(x) + h y'(x) + \frac{h^2}{2} y''(x) + \frac{h^3}{6} y'''(x) + O(h^4) \\ \frac{1}{h}(y(x + h) - y(x))) &= y'(x) + \frac{h}{2} y''(x) + \frac{h^2}{6} y'''(x) + O(h^3) \end{aligned}$$

Now note that $y'(x) = f(x,y)$, and the other derivatives of y can be put in terms of f as well.

$$\begin{aligned} y''(x) &= f_x(x,y) f_y(x,y) y'(x) = f_x(x,y) + f_y(x,y) f(x,y) = f_x + f_y f \\ y'''(x) &= f_{xx} + f_{xy} f + f_y(f_x + f_y f) + f(f_{yx} + f_{yy} f) \\ &= f_{xx} + 2f f_{xy} + f_x f_y + f f_y^2 + f^2 f_{yy} \end{aligned}$$

Therefore

$$\frac{1}{h}(y(x+h) - y(x))) = f + \frac{h}{2}(f_x + f_y f) + \frac{h^2}{6}\Big(f_{xx} + 2f f_{xy} + f_x f_y + f f_y^2 + f^2 f_{yy}\Big) + O(h^3)$$

Finally the Taylor expansion of the local truncation error can be examined.

$$\begin{aligned} T(x,y;h) &= (\alpha_1 + \alpha_2) f + \mu\alpha_2(f_x + f f_y)h + \frac{1}{2}\alpha_2\mu^2\Big(f_{xx} + 2f f_{xy} + f^2 f_{yy}\Big)h^2 \\ &\quad - \left(f + \frac{h}{2}(f_x + f_y f) + \frac{h^2}{6}\Big(f_{xx} + 2f f_{xy} + f_x f_y + f f_y^2 + f^2 f_{yy}\Big)\right) + O(h^3) \\ &= (\alpha_1 + \alpha_2 - 1) f + \left(\mu\alpha_2 - \frac{1}{2}\right)(f_x + f f_y)h \\ &\quad + \left(\left(\frac{1}{2}\alpha_2\mu^2 - \frac{1}{6}\right)\Big(f_{xx} + 2f f_{xy} + f^2 f_{yy}\Big) - \frac{1}{6}\Big(f_x f_y + f f_y^2\Big)\right)h^2 + O(h^3) \end{aligned}$$

For any general two-stage Runge-Kutta Method, $(\alpha_1 + \alpha_2 - 1) = 0$ and $\left(\mu\alpha_2 - \frac{1}{2}\right) = 0$. This implies that $\mu = \frac{1}{2\alpha_2}$. Therefore the principle error function for any general two-stage Runge-Kutta method is

$$\tau(x,y) = \left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\Big(f_{xx} + 2f f_{xy} + f^2 f_{yy}\Big) - \frac{1}{6}\Big(f_x f_y + f f_y^2\Big)$$

(b) Compare the local accuracy of the modified Euler method with that of Heun's method.
For this specific ordinary differential equation, $f(x,y) = y^\lambda$. Thus

$$\begin{aligned} f_x &= 0 \\ f_{xx} &= 0 \\ f_{xy} &= 0 \\ f_y &= \lambda y^{\lambda-1} \\ f_{yy} &= \Big(\lambda^2 - \lambda\Big) y^{\lambda-2} \end{aligned}$$

Therefore the principle error function becomes

$$\begin{aligned} \tau(x,y) &= \left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\Big(y^{2\lambda}\Big(\lambda^2 - \lambda\Big)y^{\lambda-2}\Big) - \frac{1}{6}\Big(y^\lambda \lambda^2 y^{2\lambda-2}\Big) \\ &= \left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\Big(\Big(\lambda^2 - \lambda\Big)y^{3\lambda-2}\Big) - \frac{1}{6}\Big(\lambda^2 y^{3\lambda-2}\Big) \\ &= \left(\left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\Big(\lambda^2 - \lambda\Big) - \frac{1}{6}\lambda^2\right) y^{3\lambda-2} \end{aligned}$$

For the improved Euler method, $\alpha_2 = 1$. Therefore the principle error function for the Euler method, τ_E is

$$\tau_E(x,y) = \left(\left(\frac{1}{8} - \frac{1}{6}\right)\Big(\lambda^2 - \lambda\Big) - \frac{1}{6}\lambda^2\right) y^{3\lambda-2}$$

$$= -\frac{1}{24}\Big(5\lambda^2 - \lambda\Big)y^{3\lambda-2}$$

For Heun's method, $\alpha_2 = \frac{1}{2}$. Therefore the principle error function for Heun's method, τ_H is

$$\begin{aligned}\tau_H(x,y) &= \left(\left(\frac{1}{4} - \frac{1}{6}\right)\Big(\lambda^2 - \lambda\Big) - \frac{1}{6}\lambda^2\right)y^{3\lambda-2}\\ &= -\frac{1}{12}\Big(\lambda^2 + \lambda\Big)y^{3\lambda-2}\end{aligned}$$

For what values of λ is the magnitude of the principle error function less Euler's method than Heun's method. For what values of λ is $|\tau_E| < |\tau_H|$

$$\begin{aligned}|\tau_E(x,y)| &< |\tau_H(x,y)|\\ \left|-\frac{1}{24}\Big(5\lambda^2 - \lambda\Big)y^{3\lambda-2}\right| &< \left|-\frac{1}{12}\Big(\lambda^2 + \lambda\Big)y^{3\lambda-2}\right|\\ \frac{1}{24}\Big|5\lambda^2 - \lambda\Big| &< \frac{1}{12}\Big|\lambda^2 + \lambda\Big|\\ \Big|5\lambda^2 - \lambda\Big| &< 2\Big|\lambda^2 + \lambda\Big|\\ |\lambda(5\lambda - 1)| &< |\lambda(2\lambda + 2)|\end{aligned}$$

Clearly $|\lambda(5\lambda - 1)| = |\lambda(2\lambda + 2)|$, when $\lambda = 0$. It is also equal when $(5\lambda - 1) = (2\lambda + 2)$, which implies that $\lambda = 1$. These are the only two points of intersection. When $\lambda = 2$, $|\lambda(5\lambda - 1)| > |\lambda(2\lambda + 2)|$ and when $\lambda = \frac{1}{2}$, $|\lambda(5\lambda - 1)| < |\lambda(2\lambda + 2)|$. Therefore $|\tau_E(x,y)| < |\tau_H(x,y)|$ on $\lambda \in (0,1)$, and $|\tau_H(x,y)| < |\tau_E(x,y)|$ on $\lambda \in (1,\infty)$.

(c) Determine an interval of λ such that for each λ in this interval there exists a two-stage explicit Runge-Kutta method of order $p = 3$ having parameters $0 < \alpha_1 < 1$, $0 < \alpha_2 < 1$ and $0 < \mu < 1$.

In order for a two stage explicit Runge-Kutta method to have order $p = 3$, the principle error function, $\tau(x,y)$, must be zero.

We have previously determined that $\alpha_1 = 1 - \alpha_2$ and $\mu = \frac{1}{2\alpha_2}$. Therefore for $0 < \alpha_1 < 1$, then $0 < \alpha_2 < 1$. Also for $0 < \mu < 1$, then $0 < \frac{1}{2\alpha_2} < 1$ which implies that $\frac{1}{2} < \alpha_2 < \infty$. Therefore if $\frac{1}{2} < \alpha_2 < 1$, all three conditions will be met.

In order for $\tau(x,y) = 0$,

$$\begin{aligned}0 &= \left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\Big(\lambda^2 - \lambda\Big) - \frac{1}{6}\lambda^2\\ 0 &= \left(\frac{1}{8\alpha_2} - \frac{1}{3}\right)\lambda^2 - \left(\frac{1}{8\alpha_2} - \frac{1}{6}\right)\lambda\\ 0 &= (3 - 8\alpha_2)\lambda - 3 + 4\alpha_2\\ \frac{3 - 4\alpha_2}{3 - 8\alpha_2} &= \lambda\end{aligned}$$

If $\frac{1}{2} < \alpha_2 < 1$, then $-1 < \lambda < \frac{1}{5}$. Since $\lambda > 0$, then for $0 < \lambda < \frac{1}{5}$ there exists an explicit two-stage Runge-Kutta method with order $p = 3$ and with parameters between 0 and 1.

#2 Let $\mathbf{f}(x,\mathbf{y})$ satisfy a Lipschitz condition in $\mathbf{y}$ on $[a,b] \times \mathbb{R}^d$, with Lipschitz constant L.

(a) Show that the increment function $\mathbf{\Phi}$ of the second order Runge-Kutta method

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{f}(x, \mathbf{y}) \\ \mathbf{k}_2 &= \mathbf{f}(x + h, \mathbf{y} + h\mathbf{k}_1) \\ \mathbf{\Phi}(x, \mathbf{y}; h) &= \frac{1}{2}(\mathbf{k}_1 + \mathbf{k}_2)\end{aligned}$$

also satisfies a Lipschitz condition whenever $x + h \in [a, b]$ and determine a respective Lipschitz constant M.

To show that $\mathbf{\Phi}(x, \mathbf{y}; h)$ satisfies a Lipschitz condition the value of $\|\mathbf{\Phi}(x, \mathbf{y}; h) - \mathbf{\Phi}(x, \mathbf{y}^*; h)\|$ must be shown to be bounded by a multiple of $\|y - y^*\|$. For notational simplicity, I will define the following values

$$\begin{aligned}\mathbf{k}_1^* &= \mathbf{f}(x, \mathbf{y}^*) \\ \mathbf{k}_2^* &= \mathbf{f}(x + h, \mathbf{y}^* + h\mathbf{k}_1^*) \\ \mathbf{\Phi} &= \mathbf{\Phi}(x, \mathbf{y}; h) \\ \mathbf{\Phi}^* &= \mathbf{\Phi}(x, \mathbf{y}^*; h)\end{aligned}$$

Then

$$\begin{aligned}\|\mathbf{\Phi} - \mathbf{\Phi}^*\| &= \frac{1}{2}\|\mathbf{k}_1 + \mathbf{k}_2 - \mathbf{k}_1^* - \mathbf{k}_2^*\| \\ &\leq \frac{1}{2}(\|\mathbf{k}_1 - \mathbf{k}_1^*\| + \|\mathbf{k}_2 - \mathbf{k}_2^*\|)\end{aligned}$$

Now consider $\|\mathbf{k}_1 - \mathbf{k}_1^*\|$

$$\|\mathbf{k}_1 - \mathbf{k}_1^*\| = \|\mathbf{f}(x, \mathbf{y}) - \mathbf{f}(x, \mathbf{y}^*)\|$$

Since f satisfies the Lipschitz condition

$$\|\mathbf{k}_1 - \mathbf{k}_1^*\| \leq L\|\mathbf{y} - \mathbf{y}^*\|$$

Next consider $\|\mathbf{k}_2 - \mathbf{k}_2^*\|$

$$\|\mathbf{k}_2 - \mathbf{k}_2^*\| = \|\mathbf{f}(x + h, \mathbf{y} + h\mathbf{k}_1) - \mathbf{f}(x + h, \mathbf{y}^* + h\mathbf{k}_1^*)\|$$

Since f satisfies the Lipschitz condition

$$\begin{aligned}\|\mathbf{k}_2 - \mathbf{k}_2^*\| &\leq L\|\mathbf{y} + h\mathbf{k}_1 - \mathbf{y}^* - h\mathbf{k}_1^*\| \\ \|\mathbf{k}_2 - \mathbf{k}_2^*\| &\leq L(\|\mathbf{y} - \mathbf{y}^*\| + h\|\mathbf{k}_1 - \mathbf{k}_1^*\|)\end{aligned}$$

We have already shown that $\|\mathbf{k}_1 - \mathbf{k}_1^*\| \leq L\|\mathbf{y} - \mathbf{y}^*\|$

$$\|\mathbf{k}_2 - \mathbf{k}_2^*\| \leq \left(L + hL^2\right)\|\mathbf{y} - \mathbf{y}^*\|$$

Therefore

$$\|\mathbf{\Phi} - \mathbf{\Phi}^*\| \leq \left(L + \frac{h}{2}L^2\right)\|\mathbf{y} - \mathbf{y}^*\|$$

Therefore $\mathbf{\Phi}$ satisfies a Lipschitz condition and has Lipschitz constant, $M = L + \frac{h}{2}L^2$.

(b) Show that the classical fourth order Runge-Kutta method satisifies a Lipschitz condition.

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(x, \mathbf{y}) \\
\mathbf{k}_2 &= \mathbf{f}(x + \frac{1}{2}h, \mathbf{y} + \frac{1}{2}h\mathbf{k}_1) \\
\mathbf{k}_3 &= \mathbf{f}(x + \frac{1}{2}h, \mathbf{y} + \frac{1}{2}h\mathbf{k}_2) \\
\mathbf{k}_4 &= \mathbf{f}(x + h, \mathbf{y} + h\mathbf{k}_3) \\
\mathbf{\Phi}(x, \mathbf{y}; h) &= \frac{1}{6}\mathbf{k}_1 + \frac{1}{3}\mathbf{k}_2 + \frac{1}{3}\mathbf{k}_3 + \frac{1}{6}\mathbf{k}_4
\end{aligned}$$

$$\|\mathbf{\Phi} - \mathbf{\Phi}^*\| \leq \frac{1}{6}\|\mathbf{k}_1 - \mathbf{k}_1^*\| + \frac{1}{3}\|\mathbf{k}_2 - \mathbf{k}_2^*\| + \frac{1}{3}\|\mathbf{k}_3 - \mathbf{k}_3^*\| + \frac{1}{6}\|\mathbf{k}_4 - \mathbf{k}_4^*\|$$

Now consider each of these norms individually

$$\begin{aligned}
\|\mathbf{k}_1 - \mathbf{k}_1^*\| &= \|\mathbf{f}(x, \mathbf{y}) - \mathbf{f}(x, \mathbf{y}^*)\| \\
&\leq L\|\mathbf{y} - \mathbf{y}^*\| \\
\|\mathbf{k}_2 - \mathbf{k}_2^*\| &= \left\|\mathbf{f}(x + \frac{1}{2}h, \mathbf{y} + \frac{1}{2}h\mathbf{k}_1) - \mathbf{f}(x + \frac{1}{2}h, \mathbf{y}^* + \frac{1}{2}h\mathbf{k}_1^*)\right\| \\
&\leq L\left\|\mathbf{y} + \frac{1}{2}h\mathbf{k}_1 - \mathbf{y}^* - \frac{1}{2}h\mathbf{k}_1^*\right\| \\
&\leq L\|\mathbf{y} - \mathbf{y}^*\| + \frac{1}{2}hL\|\mathbf{k}_1 - \mathbf{k}_1^*\| \\
&\leq (L + \frac{1}{2}hL^2)\|\mathbf{y} - \mathbf{y}^*\| \\
\|\mathbf{k}_3 - \mathbf{k}_3^*\| &= \left\|\mathbf{f}(x + \frac{1}{2}h, \mathbf{y} + \frac{1}{2}h\mathbf{k}_2) - \mathbf{f}(x + \frac{1}{2}h, \mathbf{y}^* + \frac{1}{2}h\mathbf{k}_2^*)\right\| \\
&\leq L\left\|\mathbf{y} + \frac{1}{2}h\mathbf{k}_2 - \mathbf{y}^* - \frac{1}{2}h\mathbf{k}_2^*\right\| \\
&\leq L\|\mathbf{y} - \mathbf{y}^*\| + \frac{1}{2}hL\|\mathbf{k}_2 - \mathbf{k}_2^*\| \\
&\leq \left(L + \frac{1}{2}hL^2 + \frac{1}{4}h^2L^3\right)\|\mathbf{y} - \mathbf{y}^*\| \\
\|\mathbf{k}_4 - \mathbf{k}_4^*\| &= \|\mathbf{f}(x + h, \mathbf{y} + h\mathbf{k}_3) - \mathbf{f}(x + h, \mathbf{y}^* + h\mathbf{k}_3^*)\| \\
&\leq L\|\mathbf{y} + h\mathbf{k}_3 - \mathbf{y}^* - h\mathbf{k}_3^*\| \\
&\leq L\|\mathbf{y} - \mathbf{y}\| + hL\|\mathbf{k}_3 - \mathbf{k}_3^*\| \\
&\leq L\|\mathbf{y} - \mathbf{y}^*\| + \left(hL^2 + \frac{1}{2}h^2L^3 + \frac{1}{4}h^3L^4\right)\|\mathbf{y} - \mathbf{y}^*\| \\
&= \left(L + hL^2 + \frac{1}{2}h^2L^3 + \frac{1}{4}h^3L^4\right)\|\mathbf{y} - \mathbf{y}^*\|
\end{aligned}$$

Let $M_1 = L$, $M_2 = L + \frac{1}{2}hL^2$, $M_3 = L + \frac{1}{2}hL^2 + \frac{1}{4}h^2L^3$ and $M_4 = L + hL^2 + \frac{1}{2}h^2L^3 + \frac{1}{4}h^3L^4$. Then

$$\|\mathbf{\Phi} - \mathbf{\Phi}^*\| \leq \left(\frac{1}{6}M_1 + \frac{1}{3}M_2 + \frac{1}{3}M_3 + \frac{1}{6}M_4\right)\|\mathbf{y} - \mathbf{y}^*\|$$

Thus $\mathbf{\Phi}$ does satisfy a Lipschitz condition and has a Lipschitz constant of $M = \frac{1}{6}M_1 + \frac{1}{3}M_2 + \frac{1}{3}M_3 + \frac{1}{6}M_4$.

(c) Show that $\mathbf{\Phi}$ for a general implicit Runge-Kutta method satisfies a Lipschitz condition.
For a general implicit Runge-Kutta method, $\mathbf{\Phi}(x, \mathbf{y}; h) = \sum_{s=1}^{r} (\alpha_s \mathbf{k}_s)$, where $\mathbf{k}_s = f(x + \mu_s h, \mathbf{y} + h\sum_{j=1}^{r} (\lambda_{sj}\mathbf{k}_j))$. I will continue to use the previously established notation.

$$\begin{aligned}
\|\mathbf{\Phi} - \mathbf{\Phi}^*\| &= \left\| \sum_{s=1}^{r} (\alpha_s \mathbf{k}_s) - \sum_{s=1}^{r} (\alpha_s \mathbf{k}_s^*) \right\| \\
&\leq \sum_{s=1}^{r} (\alpha_s \|\mathbf{k}_s - \mathbf{k}_s^*\|)
\end{aligned}$$

Now consider a single value of $\|\mathbf{k}_s - \mathbf{k}_s^*\|$

$$\|\mathbf{k}_s - \mathbf{k}_s^*\| = \left\| f(x + \mu_s h, \mathbf{y} + h\sum_{j=1}^{r} (\lambda_{sj}\mathbf{k}_j)) - f(x + \mu_s h, \mathbf{y}^* + h\sum_{j=1}^{r} \left(\lambda_{sj}\mathbf{k}_j^*\right)) \right\|$$

Since f satisfies a Lipschitz condition

$$\begin{aligned}
\|\mathbf{k}_s - \mathbf{k}_s^*\| &\leq L \left\| \mathbf{y} + h\sum_{j=1}^{r} (\lambda_{sj}\mathbf{k}_j) - \mathbf{y}^* - h\sum_{j=1}^{r} \left(\lambda_{sj}\mathbf{k}_j^*\right) \right\| \\
&\leq L\|\mathbf{y} - \mathbf{y}^*\| + hL \left\| \sum_{j=1}^{r} (\lambda_{sj}\mathbf{k}_j) - \sum_{j=1}^{r} \left(\lambda_{sj}\mathbf{k}_j^*\right) \right\|
\end{aligned}$$

Let Γ be the max of λ_{sj} for $s, j = 0, \ldots, r$

$$\|\mathbf{k}_s - \mathbf{k}_s^*\| \leq L\|\mathbf{y} - \mathbf{y}^*\| + hL\Gamma \sum_{j=1}^{r} \left(\left\| \mathbf{k}_j \mathbf{k}_j^* \right\| \right)$$

Summing both side from $s = 1$ to r results in

$$\begin{aligned}
\sum_{s=1}^{r} (\|\mathbf{k}_s - \mathbf{k}_s^*\|) &\leq sL\|\mathbf{y} - \mathbf{y}^*\| + shL\Gamma \sum_{j=1}^{r} \left(\left\| \mathbf{k}_j \mathbf{k}_j^* \right\| \right) \\
\sum_{s=1}^{r} (\|\mathbf{k}_s - \mathbf{k}_s^*\|) &\leq \frac{sL}{1 - shL\Gamma} \|\mathbf{y} - \mathbf{y}^*\|
\end{aligned}$$

Now consider $\|\mathbf{\Phi} - \mathbf{\Phi}^*\|$, and let A be the max of α_s for $s = 1, \ldots, n$

$$\begin{aligned}
\|\mathbf{\Phi} - \mathbf{\Phi}^*\| &\leq A \sum_{s=1}^{r} (\|\mathbf{k}_s - \mathbf{k}_s^*\|) \\
&\leq \frac{AsL}{1 - shL\Gamma} \|\mathbf{y} - \mathbf{y}^*\|
\end{aligned}$$

Therefore $\mathbf{\Phi}$ does satisfy a Lipschitz condition and has a Lipschitz constant of $\frac{AsL}{1-shL\Gamma}$.

#3 Consider $y' = \lambda y$ on $[0, \infty)$ for complex λ with $\mathrm{Re}(\lambda) < 0$. Let $\{u_n\}$ be the approximations of $\{y(x_n)\}$ obtained by the classical fourth-order Runge-Kutta method with the step h held constant.

(a) Show that $y(x) \to 0$ as $x \to \infty$, for any initial value y_0.
This ODE can be solved exactly for any initial condition y_0. The exact solution is $y(x) = y_0 e^{\lambda x}$. This is equivalent to $y(x) = y_0 e^{\mathrm{Re}\,\lambda x} e^{\mathrm{Im}\,\lambda i x}$. For all x and all λ, $\left| e^{\mathrm{Im}\,\lambda i x} \right| = 1$. Also since $\mathrm{Re}(\lambda) < 0$, $e^{\mathrm{Re}\,\lambda x} \to 0$ as $x \to \infty$, then $y(x) \to 0$ as $x \to \infty$.

(b) Under what condition on h can we assert that $u_n \to 0$ as $n \to \infty$? In particular what is the condition if λ is real.

To determine if $u_n \to 0$ as $n \to \infty$, the function $\phi(x, y; h)$ must be considered, because $u_n = \phi(x, y; h)u_{n-1}$. That is if $|\phi(x, y; h)| < 1$, then $u_n \to 0$ when $n \to \infty$.

$$
\begin{aligned}
\phi(h\lambda)y &= y + h\Phi(x, y; h) \\
\Phi(x, y; h) &= \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \\
k_1 &= f(x, y) \\
&= \lambda y \\
k_2 &= f(x + \frac{1}{2}h, y + \frac{1}{2}hk_1) \\
&= \lambda(y + \frac{1}{2}h\lambda y) \\
&= \lambda y + \frac{1}{2}h\lambda^2 y \\
&= (\lambda + \frac{1}{2}h\lambda^2)y \\
k_3 &= f(x + \frac{1}{2}h, y + \frac{1}{2}hk_2) \\
&= \lambda(y + \frac{1}{2}hk_2) \\
&= \lambda(y + \frac{1}{2}h(\lambda y + \frac{1}{2}h\lambda^2 y)) \\
&= \lambda y + \frac{1}{2}h\lambda^2 y + \frac{1}{4}h^2\lambda^3 y \\
&= (\lambda + \frac{1}{2}h\lambda^2 + \frac{1}{4}h^2\lambda^3)y \\
k_4 &= f(x + h, y + hk_3) \\
&= \lambda(y + h(\lambda + \frac{1}{2}h\lambda^2 + \frac{1}{4}h^2\lambda^3)y) \\
&= (\lambda + h\lambda^2 + \frac{1}{2}h^2\lambda^3 + \frac{1}{4}h^3\lambda^4)y \\
\Phi(x, y; h) &= \left(\frac{1}{6}\lambda + \frac{1}{3}(\lambda + \frac{1}{2}h\lambda^2) + \frac{1}{3}(\lambda + \frac{1}{2}h\lambda^2 + \frac{1}{4}h^2\lambda^3) + \frac{1}{6}(\lambda + h\lambda^2 + \frac{1}{2}h^2\lambda^3 + \frac{1}{4}h^3\lambda^4)\right)y \\
&= \left(\lambda + \frac{1}{2}h\lambda^2 + \frac{1}{6}h^2\lambda^3 + \frac{1}{24}h^3\lambda^4\right)y \\
\phi(h\lambda) &= 1 + h\lambda + \frac{1}{2}h^2\lambda^2 + \frac{1}{6}h^3\lambda^3 + \frac{1}{24}h^4\lambda^4\phi(z) \qquad = 1 + z + \frac{1}{2}
\end{aligned}
$$

This in order for $|\phi(z)| < 1$, $\left|1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4\right| < 1$ This cannot be solved exactly easily. Using Mathematica is can be found that if $-2.7859 < z < 0$, then $|\phi(z)| < 1$ Therefore for $u_n \to 0$ as $n \to \infty$, $0 < h < -2.7859/\lambda$.

(c) What is the analogous result for Euler's method.

For Euler's method

$$
\begin{aligned}
\Phi(x, y; h) &= f(x, y) \\
&= \lambda y\phi(h\lambda) \qquad = 1 + h\lambda \\
\phi(z) &= 1 + z
\end{aligned}
$$

Thus in order for $|\phi(z)| < 1$, $-2 < z < 0$. Therefore for $u_n \to 0$ as $n \to \infty$, $0 < h < -2/\lambda$.

(d) Generalize to a system $\mathbf{y}' = A\mathbf{y}$, where A is a constant matrix with eigenvalues with negative real parts.

For this generalized system, the equations of $\phi(z)$ stay the same, and we are considering hA instead of $h\lambda$. Also note that using recursion $u_n = \phi(hA)^n u_0$. Therefore for $u_n \to 0$ as $n \to \infty$, $|\phi(hA)^n| \to 0$ as $n \to \infty$. In order for the matrix $|\phi(hA)^n|$ to converge to the zero matrix, the spectral radius of $\phi(hA)$, $\rho(\phi(hA))$ must be less than one. The spectral radius is the maximum of the absolute values of the eigenvalues. Thus for $u_n \to \infty$ as $n \to \infty$, $\rho(\phi(hA)) < 1$. Also it is known that since ϕ is a polynomial, $\rho(\phi(hA)) = |\phi(h\rho(A))|$

For the classical fourth-order Runge-Kutta method, we have already shown that $|\phi(h\lambda)| < 1$ implies that $0 < h < \frac{-2.7859}{\lambda}$. Therefore for $|\phi(h\rho(A))| < 1$, $0 < h < \frac{-2.7859}{\rho(A)}$. Since for all eigenvalues, λ_i of A, $|\lambda_i| \le \rho(A)$, this implies that $0 < h < \frac{-2.7859}{\lambda_i}$ for all eigenvalues of A.

Similarly for Euler's method, $0 < h < \frac{-2}{\rho(A)} < \frac{-2}{\lambda_i}$. Thus the value of h is related to the spectral radius of A.

#4 Consider the linear homogeneous system

$$\mathbf{y}' = A\mathbf{y},\ y \in \mathbb{R}^d$$

with constant coefficient matrix $A \in \mathbb{R}^{d\times d}$

(a) For Euler method applied to this system, determine $\phi(z)$ and the principle error function.

We have previously determined in problem 3, that $\phi(z) = 1 + z$ for this method on this system. In order to find the principe error function the local truncation error must be computed. The local truncation error is given by $T(x, \mathbf{y}; h) = \frac{1}{h}\Big(\phi(hA) - e^{hA}\Big)\mathbf{y}$

$$T(x, \mathbf{y}; h) = \frac{1}{h}\Big(1 + hA - e^{hA}\Big)\mathbf{y}$$

Taking the Taylor expansion of e^{hA}

$$\begin{aligned} T(x, \mathbf{y}; h) &= \frac{1}{h}\Big(1 + hA - 1 - hA - \frac{1}{2}(hA)^2 - O((hA)^3)\Big)\mathbf{y} \\ &= \Big(-\frac{1}{2}hA^2 - O(h^2)\Big)\mathbf{y} \end{aligned}$$

Therefore Euler's method is of order 1 and has principle error function

$$\tau(x, y) = -\frac{1}{2}A^2\mathbf{y}$$

(b) Do the same for the classical fourth-order Runge-Kutta method.

We have previously shown in problem 3, that $\phi(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{3!}z^3 + \frac{1}{4!}z^4$, for this problem and the classical fourth-order Runge-Kutta's method. Again to find the principle error function the local truncation error must be computed.

$$T(x, \mathbf{y}; h) = \frac{1}{h}\Big(1 + hA + \frac{1}{2}(hA)^2 + \frac{1}{3!}(hA)^3 + \frac{1}{4!}(hA)^4 - e^{hA}\Big)\mathbf{y}$$

Taking the Taylor expansion of e^{hA} and canceling terms results in

$$T(x, \mathbf{y}; h) = \frac{1}{h}\Big(-\frac{1}{5!}h^5A^5 + O(h^6)\Big)\mathbf{y} \qquad = \Big(\frac{1}{5!}h^4A^5 + O(h^5)\Big)\mathbf{y}$$

Therefore the classical fourth-order Runge-Kutta method has order 4 and principle error function

$$\tau(x, \mathbf{y}) = \frac{1}{120}A^5\mathbf{y}$$

#5 (a) I created a set of objects to represent different methods of solving a system of ODEs The most general object represents explicit one-step methods and is able to apply to method to approximate the system with the function solveSystem.m. Look specifically as the functions solveSystem.m and phi.m to see the inner workings of this system.

```
classdef (Abstract) explicitOneStepMethod
%EXPLICITONESTEPMETHOD - Abstract class to represent the generic way to solve
%a system of ODEs via an explicit one step method
%
%A method is one step if it only considers the information from the previous step
%A method is explicit if the next step can be found using current information
%An implicit method generally requires solving an system of equations
%
% This is an abstract class and cannot be instantiated
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 16-November-2015

    methods (Abstract)
        phi = phi(f, x, y, h);
    end

    methods
        y = solveSystem(ExplicitOneStepMethod, f, x, yInit);
    end

end
```

```
function [y] = solveSystem(explicitOneStepMethod, f, x, yInit)
%SOLVESYSTEM - function to apply an explicit one step method to a
%system of ODEs and find a numerical solution
%This function is part of the explicitOneStepMethod Class
%As such the oject this method is attached to is passed in as first argument
%
% Where Phi is an NumericalAnalysis explicitOneStepMethod object
% Syntax:  [y] = Phi.solveSystem(f, x, yInit)
%
% Inputs:
%    f - function describing system of ODEs, must accept as input vector the
%       size of yInit
%    x - set of real numbers that the
%    yInit - the initial value of the system at x(1);
%
% Outputs:
%    y - matrix whose columns are points found numerically to approximate
%       the solution to the system of ODEs
%
% Example: %TODO add example
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
```

```
% See also:

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 15-November-2015
    p = inputParser;
    p.addRequired('f', @Utils.isFunctionHandle);
    p.addRequired('x', @Utils.isGridFunction);
    p.addRequired('yInit', @Utils.isNumericVector);
    p.parse(f, x, yInit);

    % find all of the step sizes
    h = diff(x);
    n = length(x);

    % set up matrix to store solution
    y = zeros(length(yInit), n);
    y(:, 1) = yInit;

    for i = 1:n-1
        % find the next value of y
        y(:, i+1) = y(:,i) + h(i)*explicitOneStepMethod.phi(f, x(i), y(:,i), h(i));
    end
end
```

I created another object to be able to represent any explicit Runge-Kutta method, and implemented the Φ method.

```
classdef explicitRungeKuttaMethod < NumericalAnalysis.ODES.explicitOneStepMethod
%EXPLICITRUNGEKUTTAMETHOD - The EulerMethod class represents the euler one step ...
    method for
%numerically approximating the solution of a system of ODES
%The Euler method is of order 1 and is based on the forward difference
%approximation of the derivative
%
% Syntax:  rk = NumericalAnalysis.ODES.ExplicitRungeKuttaMethod()
%
% Inputs
%    alpha - weights of average of each stage
%    lambda - weights to find each stage based on previous stages
%
% Example:
%    % Heun's Method Example
%    alpha = [1/4, 0, 3/4];
%    lambda = [0, 0, 0; 1/3, 0, 0; 0, 2/3, 0];
%    heun = NumericalAnalysis.ODES.ExplicitRungeKuttaMethod(alpha, lambda);
%    % now use heun to solve you system of ODES with heun.solveSystem
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
% See also: EXPLICITONESTEPMETHOD

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 16-November-2015
    properties
        % positive integer represents the number of stages
```

```
        r

        % coefficients of weights for average
        % stored as row vector
        alpha

        % weights of previous stages used to determine next stage
        lambda

        % sum of rows of lambda
        % represents size of step to next stage
        % stored as column vector
        mu
    end

    methods
        function obj = explicitRungeKuttaMethod(alpha, lambda)
            p = inputParser();
            p.addRequired('alpha', @Utils.isNumericVector);
            p.addRequired('lambda', @(x) Utils.isLowerTriangular(x) && ...
                Utils.isSquareMatrix(x));
            p.parse(alpha, lambda);

            % check inputs further
            % for a consistent method sum of alpha must be one
            if(abs(sum(alpha) - 1) > 10*eps)
                error('For a consistent RungeKutta Method the sum of the alphas ...
                    must be one');
            end

            % length of alpha must be the same as size of lambda
            if(length(alpha) ≠ length(lambda))
                error('The size of alpha and lambda must agree');
            end

            % to be explicit method lambda must be lower triangular with zeros
            % on diagonal
            if(any(diag(lambda)))
                error('For explicit RungeKutta method, the diagonal of lambda ...
                    must be zeros');
            end

            % make sure alpha is row vector
            if(iscolumn(alpha))
                alpha = alpha.';
            end

            obj.r = length(alpha);
            obj.alpha = alpha;
            obj.lambda = lambda;
            obj.mu = sum(lambda, 2);
        end

        phi = phi(ExplicitRungeKuttaMethod, f, x, y, h);
    end
end
```

```
function [phi] = phi(explicitRungeKuttaMethod, f, x, y, h)
%PHI - calculates phi for the ExplicitRungeKuttaMethod class
%
```

```
% Syntax:  phi = rk.phi(f, x, y, h)
% rk is a NumericalAnalysis.ODES.ExplicitRungeKuttaMethod object
%
% Inputs:
%    f - function that defines the system of ODEs
%    x - number representing current place on grid function
%    y - current value of system at x
%    h - step size to next value of x
%
% Outputs:
%    phi - next value of y is calculated as y + h*phi
%
% Example:
%    % alpha and lambda for Heun's method
%    alpha = [1/4, 0, 3/14];
%    lambda = [0, 0, 0; 1/3, 0, 0; 0, 2/3, 0];
%    rk = NumericalAnalysis.ODES.ExplicitRungeKuttaMethod(alpha, lambda);
%    f = @(x, y) x*y;
%    x = 1;
%    y = 2;
%    h = 1;
%    phi = rk.phi(f, x, y, h);
%    yNext = y + h*phi;
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
% See also: ODEEXPLICITONESTEPMETHOD, ODEEXPLICITONESTEPMETHOD.SOLVESYSTEM

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 16-November-2015
    p = inputParser();
    p.addRequired('f', @Utils.isFunctionHandle);
    p.addRequired('x', @Utils.isNumber);
    p.addRequired('y', @Utils.isNumericVector);
    p.addRequired('h', @Utils.isNumber);
    p.parse(f, x, y, h);


    k = zeros(length(y), explicitRungeKuttaMethod.r);
    for i=1:explicitRungeKuttaMethod.r
        k(:,i) = f(x + explicitRungeKuttaMethod.mu(i)*h, y + ...
            h*k*explicitRungeKuttaMethod.lambda(i, :).');
    end
    phi = k*explicitRungeKuttaMethod.alpha.';
end
```

Lastly there are objects to actually represent Euler's method and the classical fourth-order Runge-Kutta method.

```
classdef eulerMethod < NumericalAnalysis.ODES.explicitRungeKuttaMethod
%EULERMETHOD - The eulerMethod class represents the euler one step method for
%numerically approximating the solution of a system of ODES
%The Euler method is of order 1 and is based on the forward difference
%approximation of the derivative
%The Euler method can also be viewed as a one stage RungeKutta Method
%such that alpha = 1 and lambda = mu = 0
%
```

```
% Syntax:  euler = NumericalAnalysis.ODES.eulerMethod()
%
% no inputs necessary for creation of object
%
% Example: see syntax
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
% See also: EXPLICITRUNGEKUTTAMETHOD

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 16-November-2015
    methods
        function [obj] = eulerMethod()
            alpha = [1];
            lambda = [0];
            obj@NumericalAnalysis.ODES.explicitRungeKuttaMethod(alpha, lambda);
        end
    end
end
```

```
classdef standardRK4Method < NumericalAnalysis.ODES.explicitRungeKuttaMethod
%STANDARDRK4METHOD - The standardRK4Method class represents the standard Runge
%Kutta four stage method for
%numerically approximating the solution of a system of ODES
%The standard RungeKutta four stage method is of order 4
%
% Syntax:  rk4 = NumericalAnalysis.ODES.standardRK4Method()
%
% no inputs necessary for creation of object
%
% Example: see syntax
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
% See also: EXPLICITRUNGEKUTTAMETHOD

% Author: Caleb Logemann
% email: logemann@iastate.edu
% Website: http://www.logemann.public.iastate.edu/
% November 2015; Last revision: 16-November-2015
    methods
        function [obj] = standardRK4Method()
            alpha = [1/6, 1/3, 1/3, 1/6];
            lambda = [0, 0, 0, 0; 1/2, 0, 0, 0; 0, 1/2, 0, 0; 0, 0, 1, 0];
            obj@NumericalAnalysis.ODES.explicitRungeKuttaMethod(alpha, lambda);
        end
    end
end
```

(b) This is the script that actually uses these object to solve the system

```
yInit = [1; 1; 1];
```

```
% create objects to solve ODES with specific method
euler = NumericalAnalysis.ODES.eulerMethod;
rk4 = NumericalAnalysis.ODES.standardRK4Method;

% non stiff problem
lambda1 = 1;
lambda2 = 0;
lambda3 = 1;

A = 1/2*[lambda2 + lambda3, lambda3 - lambda1, lambda2 - lambda1;
lambda3 - lambda2, lambda1 + lambda3, lambda1 - lambda2;
lambda2 - lambda3, lambda1 - lambda3, lambda1 + lambda2];

f = @(x,y) A*y;

% store actual solution as function
y = @(x) [-exp(lambda1*x) + exp(lambda2*x) + exp(lambda3*x);
exp(lambda1*x) - exp(lambda2*x) + exp(lambda3*x);
exp(lambda1*x) + exp(lambda2*x) - exp(lambda3*x)];

eulerError = [];
rkError = [];

for N=[5,10,20,40,80]
    x = linspace(0,1,N);

    % compute actual solution
    yActual = y(x);

    u = euler.solveSystem(f, x, yInit);
    eulerError = [eulerError; norm(yActual - u)];

    u = rk4.solveSystem(f, x, yInit);
    rkError = [rkError; norm(yActual-u)];
end

% display error
format long
eulerError
rkError

% stiff problem
lambda1 = 0;
lambda2 = -1;
lambda3 = -100;

A = 1/2*[lambda2 + lambda3, lambda3 - lambda1, lambda2 - lambda1;
lambda3 - lambda2, lambda1 + lambda3, lambda1 - lambda2;
lambda2 - lambda3, lambda1 - lambda3, lambda1 + lambda2];

f = @(x,y) A*y;

% store actual solution as function
y = @(x) [-exp(lambda1*x) + exp(lambda2*x) + exp(lambda3*x);
exp(lambda1*x) - exp(lambda2*x) + exp(lambda3*x);
exp(lambda1*x) + exp(lambda2*x) - exp(lambda3*x)];

eulerError = [];
rkError = [];

for N=[5,10,20,40,80]
```

```
    x = linspace(0,1,N);

    % compute actual solution
    yActual = y(x);

    u = euler.solveSystem(f, x, yInit);
    eulerError = [eulerError; norm(yActual - u)];

    u = rk4.solveSystem(f, x, yInit);
    rkError = [rkError; norm(yActual-u)];
end

% display error
format long
eulerError
rkError
```

```
eulerError =

   0.669647499864682
   0.433642339396205
   0.294149701224462
   0.203833950720079
   0.142702928842803


rkError =

   1.0e-03 *

   0.172958606434202
   0.009918694490326
   0.000715540503618
   0.000057312979472
   0.000004825933441


eulerError =

   1.0e+12 *

   0.000000575152398
   0.001922582741375
   1.643413090818388
   0.000084922547445
   0.000000000000952


rkError =

   1.0e+24 *

   0.000000065788346
```

```
1.537651043959767
0.553810514762947
0.000000000000000
0.000000000000000
```

I learned from these examples that for stiff problems even good numerical methods can become very unstable if h is not in the region of A-stabity. In the stiff problem both methods have extremely large error. It is interesting to note that the Runge-Kutta method has much larger error for this stiff problem than Euler's method. This seems to indicate that when they are both unstable the Runge-Kutta method is much more unstable. However once method is unstable the magnitude of the error is not really important. The Runge-Kutta method has a larger region of A-stability. You can see that the error drops to almost 0, once h enters the region of A-stability. A much smaller h is required for this to happen with Euler's method.

Also for the nonstiff problem, not how much smaller the error is for Runge-Kuttas method. It is also possible to see the error in Runge-Kutta's method is divided by 64 as h is cut in half. For Euler's method the error does not even decrease by half as h is cut in half. This indicates that Runge-Kutta's method is of order 4 and Euler's method is of order 1, as we previously have shown.