MA/CS 615 Spring 2019 Homework #2

Due before class starts on Feb 21. Late homework will not be given any credit. Collaboration is OK but not encouraged. Indicate on your report whether you have collaborated with others and whom you have collaborated with.

1. (20 points) Analyze the wellposedness of the initial value problem

$$u_t = \alpha u_{xx} + \beta u_{xxxx}, \quad u(x,0) = f(x)$$

for four different cases:

- $\alpha \geq 0$, $\beta \geq 0$.
- $\alpha \geq 0$, $\beta < 0$.
- $\alpha < 0$, $\beta \geq 0$.
- $\alpha < 0$, $\beta < 0$.

2. (30 pts) The explicit third order strong stability preserving (SSP) Runge-Kutta method can be written as

$$\begin{aligned} U^{(1)} &= U^n + \Delta t f(U^n), \\ U^{(2)} &= \tfrac{3}{4}U^n + \tfrac{1}{4}(U^{(1)} + \Delta t f(U^{(1)})), \\ U^{n+1} &= \tfrac{1}{3}U^n + \tfrac{2}{3}(U^{(2)} + \Delta t f(U^{(2)})). \end{aligned} \tag{0.1}$$

in which $U^{(1)}$ and $U^{(2)}$ are obviously different from those in a standard form. The strong stability here (different from what we defined for LMM) refers to the feature that U^{n+1} can be written as a convex combination of several formal forward Euler steps, which is useful in enforcing nonlinear stability for high order Runge-Kutta methods.

(a) (10 pts) Rewrite it in the standard form and find its Butcher tableau ($\mathbf{c}$ can be found as the sum of each row in A).

$\mathbf{c}$	A
	$\mathbf{b}$

(b) (10 pts) Show that its local truncation error is third order for solving $u'(t) = f(u)$ (here we consider a simpler case in which f is a function of u only).

(c) (10 pts) After eliminating inner stages, any explicit 3-stage Runge-Kutta method can be written as $U^{n+1} = R(z)U^n$ where $R(z) = 1 + z\mathbf{b}^T(I + zA + z^2A^2)\mathbf{e}$ with $z = \Delta t\lambda$ and $\mathbf{e} = [1, \cdots, 1]^T$. Verify that $R(z) = 1 + z + z^2/2 + z^3/6$ for this third order Runge-Kutta by plugging in A and $\mathbf{b}$. So this scheme has the same region of absolute stability as any other explicit 3-stage Runge-Kutta of order 3.

3. (20 pts) The explicit 4-stage fourth order Runge-Kutta for solving $u' = f(u,t)$ can also be written/implemented as

$$F^{(1)} = f(U^n, t^n)$$

$$
\begin{aligned}
F^{(2)} &= f(U^n + \frac{1}{2}\Delta t F^{(1)}, t^n + \frac{1}{2}\Delta t) \\
F^{(3)} &= f(U^n + \frac{1}{2}\Delta t F^{(2)}, t^n + \frac{1}{2}\Delta t) \\
F^{(4)} &= f(U^n + \Delta t F^{(3)}, t^n + \Delta t) \\
U^{n+1} &= U^n + \frac{1}{6}\Delta t(F^{(1)} + 2F^{(2)} + 2F^{(3)} + F^{(4)})
\end{aligned}
$$

thus it requires four extra storage $F^{(i)}$, $i = 1, 2, 3, 4$ (or $U^{(1)}$, $i = 1, 2, 3, 4$ if implemented in the standard form). In the context of solving time-dependent PDEs by an explicit ODE solver, each $F^{(i)}$ or $U^{(i)}$ could be huge (think about solving the 3D Maxwell's equation). As an alternative, a low storage explicit 5-stage fourth order Runge-Kutta (LSERK) method is given by

$$
\begin{aligned}
U^{(0)} &= U^n, V^{(0)} = U^n \\
V^{(i)} &= a_i V^{(i-1)} + \Delta t f(U^{(i-1)}, t^n + c_i \Delta t), \quad i = 1, \cdots, 5 \\
U^{(i)} &= U^{(i-1)} + b_i V^{(i)}, \quad i = 1, \cdots, 5 \\
U^{n+1} &= U^{(5)}.
\end{aligned}
$$

with the coefficients given by

```
 rk4a = [
0.0 ...
-567301805773.0/1357537059087.0 ...
-2404267990393.0/2016746695238.0 ...
-3550918686646.0/2091501179385.0 ...
-1275806237668.0/842570457699.0];
rk4b = [ 1432997174477.0/9575080441755.0 ...
5161836677717.0/13612068292357.0 ...
1720146321549.0/2090206949498.0 ...
3134564353537.0/4481467310338.0 ...
2277821191437.0/14882151754819.0];
rk4c = [
0.0 ...
1432997174477.0/9575080441755.0 ...
2526269341429.0/6820363962896.0 ...
2006345519317.0/3224310063776.0 ...
2802321613138.0/2924317926251.0];
```

In each time step, it requires to evaluate the function f five times whereraz the explicit 4-stage fourth order Runge-Kutta only needs four times. In general, evaluating a nonlinear function f (or nonlinear/linear operator f in solving PDEs) is the most computationally expensive part. However, this does not necessarily mean that LSERK is more expensive than the 4-stage fourth order RK. For instance, if LSERK has a larger stability region then it may allow larger time steps to offset extra cost in one time step.

(a) (10 pts) The LSERK solving $u' = \lambda u$ can be written as $U^{n+1} = R(z)U^n$. Find what $R(z)$ is in terms of a, b and c. (Hint: The first five terms in $R(z)$ are very easy to find

(why?). Once you know the first five terms, it becomes much simpler to find other higher order terms because you can ignore lower order terms of z when finding the coefficients for higher ones. There is no need to plug in the values of a, b and c. But we need to use the fact that $a_1 = 0$.)

(b) (10 pts) Plot the stability regions of LSERK and the 4-stage fourth order RK. Which one is larger? Suppose we want solve the semidiscrete scheme $\mathbf{U}'(t) = -\frac{1}{\Delta x^2} K\mathbf{U}$ (K is the tridiagonal $-1, 2, -1$ matrix) for the heat equation using these two Runge-Kutta methods (with the largest possible stable time step), then for a given final time $T = 1$ which Runge-Kutta is more efficient (less number of matrix-vector multiplications $K\mathbf{U}$)?

4. (30-35 pts) Consider the Kepler problem:

$$q' = H_p, \quad p' = -H_q,$$

where $p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$, $q = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$ and the Hamiltonian is given by

$$H(q,p) = \frac{p_1^2 + p_2^2}{2} - \frac{1}{r} - \frac{0.01}{2r^3}$$

and $r = \sqrt{q_1^2 + q_2^2}$. The initial conditions are

$$q_1(0) = 1 - \beta, q_2(0) = 0, p_1(0) = 0, p_2(0) = \sqrt{(1+\beta)/(1-\beta)}, \beta = 0.6.$$

Clearly, $H' = H_q q' + H_p p' = 0$, so $H(q(t), p(t)) = H(q(0), p(0))$. Thus $|H(q(t), p(t)) - H(q(0), p(0))|$ could be a simple error indicator, which *underestimates* the true error. The solution can be visualized by plotting $q_2(t)$ vs. $q_1(t)$ in the q_2-q_1 plane (phase plane portrait), which looks like curves on a torus. Download Kepler_reference_q.m and run the following script in MATLAB to plot a reference solution in Figure 0.1

```
load('Kepler_reference_q.mat');plot(q(1,:),q(2,:));
axis equal;xlabel('q1');ylabel('q2');
```

(a) Use the fourth-order Runge-Kutta to solve this ODE system with $\Delta t = 0.0005$ to the final time $T = 500$ as a reference solution.

(b) Consider the second-order accurate A-stable implicit midpoint method

$$\frac{u^{n+1} - u^n}{\Delta t} = f\left(\frac{u^{n+1} + u^n}{2}, t^{n+\frac{1}{2}}\right)$$

for the ODE $u' = f(u,t)$. Implement the midpoint method for the ODE system. Use Newton's iteration to solve the nonlinear equations.

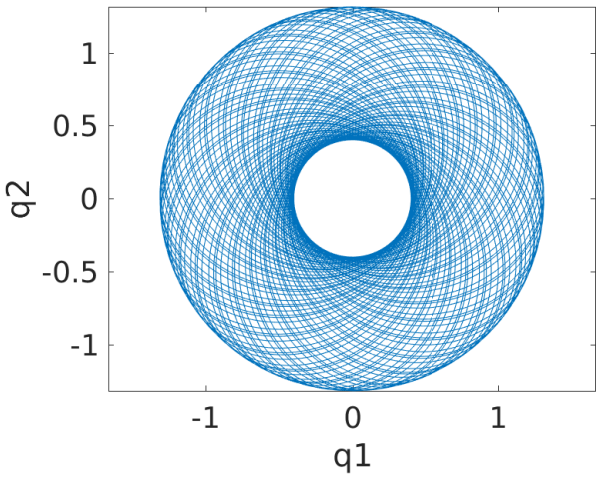

Figure 0.1: phase plane portrait

(c) (30 points) Run the fourth-order Runge-Kutta and the midpoint method with $\Delta t = 0.1$ and $\Delta t = 0.01$ to the final time $T = 500$ (four runs in total). Plot the phase plane portraits (four figures with the same scale). Monitor the maximum of $|H(q(t), p(t)) - H(q(0), p(0))|$. Let $\bar{u}(t)$ denote the reference solution ($u = (q, p)$ and you need to generate this reference using 4th order RK). Define the true error as $e(t) = |u(t) - \bar{u}(t)|$. Plot $\log e(t)$ vs. t (four curves in total).

(d) (Bonus 5 points) What are your observations and conclusions for this particular problem?

For instance, a forward Euler method to solve it can be implemented as (it will not generate an accurate solution with $\Delta t = 0.001$):

```
function Kepler_forwardEuler

T=500;
dt=0.001;
Nt=T/dt;
% u(:,i) is a vector of size 4 at time (i-1)*dt
% u(1)=q1, u(2)=q2, u(3)=p1, u(4)=p2
u=zeros(4,Nt+1);
% The initial condition
beta=0.6;
u(1,1)=1-beta;
u(4,1)=sqrt((1+beta)/(1-beta));
% forward Euler
for i=1:Nt
    u(:,i+1)=u(:,i)+dt*RHS_Kepler(u(:,i));
end
% Phase Portrait of q1-q2
plot(u(1,:),u(2,:));axis equal
```

```
xlabel('q1');ylabel('q2')
set(0,'DefaultTextFontSize',18,'DefaultAxesFontSize', 18)

 function f = RHS_Kepler(u)
% The right hand side functiosn
% H_p and -H_q for the modified Kepler problem
% u(1)=q1, u(2)=q2, u(3)=p1, u(4)=p2
   r = sqrt(u(1)^2+u(2)^2);
   f=zeros(size(u));
   f(1)=u(3);
   f(2)=u(4);
   f(3)=-u(1)/r^3-0.015*u(1)/r^5;
   f(4)=-u(2)/r^3-0.015*u(2)/r^5;
```