Quiz 1 - Computational Physics II

NAME: Juan Daniel Vásconez **SCORE:** 16/20

Date: Thursday 27 February 2025 **Duration:** 45 minutes

Credits: 20 points (4 questions) **Type of evaluation:** LAB

Provide short and concise answers **to the following items:**

1. **(5 points) Integration methods for Ordinary Differential Equations (ODEs)**
   (a) Explain the difference between explicit and implicit Euler integrators for ODEs.
   (b) What are the error sources when you integrate ODEs numerically in a computer?

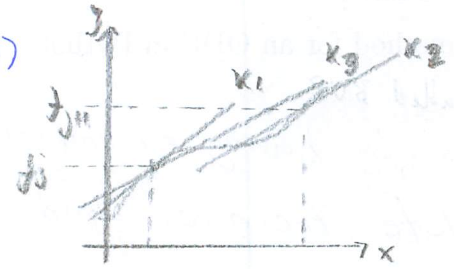
a) Explicit Euler methods depend only on the ~~main~~ ~~step~~ current state and apoximate the next one from the ~~main~~ ~~step~~ current state, Implicit Euler Integrators depend on the next ~~step~~ state itself and the main step.

-0.5

b) You can overfit or underfit? we can get erors from the machine epsilon sigma X or the truncation error, and we can understimate or overstimat by the lack or excess of free parameters

2. **(5 points) Runge-Kutta methods for ODEs**
   (a) Explain how Runge-Kutta (RK) methods work.
   (b) How do RK methods improve upon simpler integration methods like the Euler methods?
   (c) Design your own third-order RK method, and write down the slopes and integrator.

a)



Runge kutta methods Approximate an tangent slope of the function betwen 2 points having a slope for the initial and last point and calculating an average for the next order → RK2?

-1.5

b) Since RK methods is a combination of Implicit and explicit Euler methods, the apoximation is gonna improbe also for the conection on the Higher order by taking the average and minimizing the error X

Between the slopes

↳ which error?

c) What about stability?

h is missing.

$$K_1 = S(t_1) + hF(S(t_i - \tfrac{1}{3}), + t_j + \tfrac{1}{3}) \quad X$$

$-\frac{1}{3}$?

$$K_2 = S(t_j) + \frac{h}{2}F[(S(t_j - \tfrac{1}{3}) + t_j - \tfrac{1}{3}) + (S(t_j + \tfrac{1}{3}) + t_j + \tfrac{1}{3})] \quad X$$

$a_1 K_1$

$$K_3 = S(t_j) + hF(S(t_j + \tfrac{1}{3}), + t_j + \tfrac{1}{3}) \quad X$$

$c_2 K_2$

$$S(t_{j+1}) = S(t_j) + \frac{h}{4}(K_1 + 2K_2 + K_3) \quad ✓$$

3. **(5 points) ODE order reduction**

Consider an object with mass, m, that falls from rest under the influence of gravity (i.e., along the Y axis). The object is also subjected to a drag force that arises from friction with air molecules, so its equation of motion reads:

$$m\frac{d^2y}{dt^2} = -m\,g + b\frac{dy}{dt}$$

where g is the acceleration of gravity and b is a friction constant.

(a) Reduce the order of this ODE to first order and write down the resulting matrix-form equation.

(b) Identify the slope function.

(c) Briefly explain what the advantages of carrying out order reduction are.

a) $\frac{d^2y}{dt^2} = -g + \frac{b}{m}\frac{dy}{dt}$ ✓ $\quad s(t) = \begin{bmatrix} y \\ y' \end{bmatrix} \rightarrow \frac{ds(t)}{dt} = \begin{bmatrix} y' \\ y'' \end{bmatrix}$ ✓ ✓

$$\frac{ds(t)}{dt} = \begin{bmatrix} y' \\ \left(-g + \frac{b}{m}\right)\frac{dy}{dt} \end{bmatrix} \rightarrow \frac{ds(t)}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & -g + \frac{b}{m} \end{bmatrix} \begin{bmatrix} y \\ y' \end{bmatrix}$$ ✓

$s(t)$

−1

b) $\frac{d^2y}{dt^2} = \boxed{-g + \frac{b}{m}\frac{dy}{dt}} \rightarrow$ slope ✗ ↑ This is the slope function

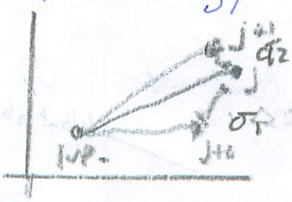
c) We can reduce Higher order ODEs to linear first order EQNs making the possiple to solve with common ODES solvers with a linear algebra form ✓

4. **(5 points) Shooting method for ODEs**

(a) Explain how the shooting method works and what it is used for.

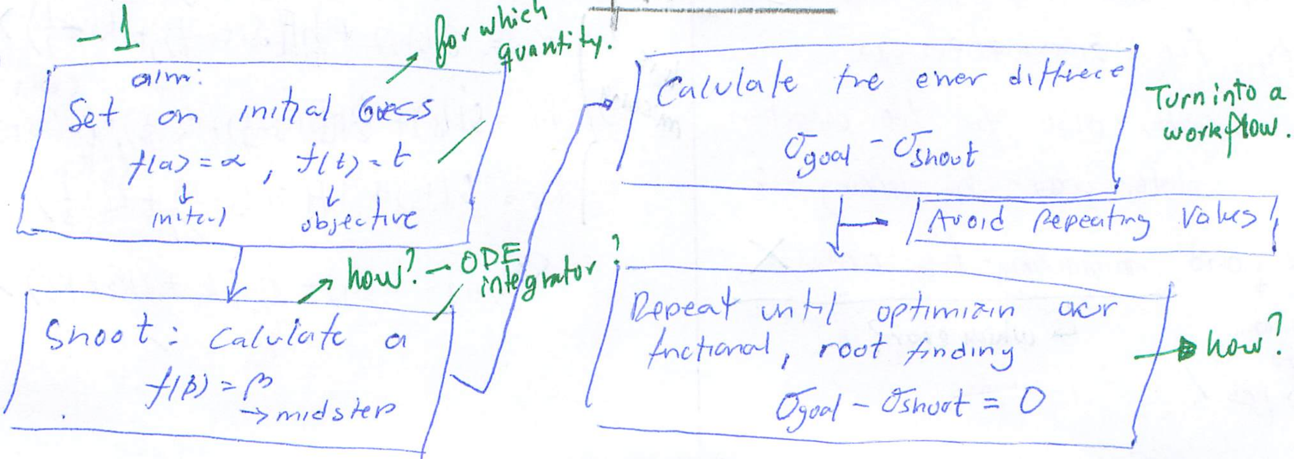
(b) Sketch an algorithm workflow to implement the shooting method for an ODE in Python.

a) It's an optimazation problem, → for the problems called BVP which start's by having an initial guess and shotting prediction for after calulate the error and minimize it. Highly computational cost commonly, and is used for aproxmating solutions



$F(0) = \Delta\sigma = 0$

b) −1



aim: Set an initial Guess ✓

$f(a) = \alpha$, $f(b) = b$

↓ initial ↓ objective

→ for which quantity.

Shoot: Calulate a $f(\beta) = \beta$ → midstep

→ how? — ODE integrator?

Calulate the error diffrece $\sigma_{goal} - \sigma_{shoot}$

Turn into a workflow.

Avoid repeating values!

Repeat until optimizin our functional, root finding $\sigma_{goal} - \sigma_{shoot} = 0$

→ how?