Midterm Exam (part 1) - Computational Physics II

NAME: Alan Palma Tavez **SCORE:** 9.5/10

Date: Tuesday 8 April 2025 **Duration:** 45 minutes
Credits: 10 points (5 questions) **Type of evaluation:** Midterm Exam

Provide concise answers to the following items.

1. **(2 points) Trapezoidal method for ordinary differential equations (ODEs)**
   (a) Explain how the trapezoidal method for solving ODEs works.

The trapezoidal method Formula is: $y_{i+1} = y_i + \frac{h}{2}\left(F(t_i, y_i) + F(t_{i+1}, s_{i+1})\right)$
What this methods does is aproximating the solution by taking two points, and the averaging their slopes, for this it is a 2nd order method.

   (b) Consider the radioactive decay ODE with decay constant, α. If there are $N(t)$ radioactive nuclei at time t and N_0 at $t = 0$, and if their rate of decay $(-dN/dt)$ is proportional to the number of undecayed nuclei, then:

$$\frac{dN}{dt} = -\alpha N$$

   Indicate the slope and explain how the trapezoidal formula would be implemented in Python.

The slope of this ODE is: $F(N,t) = -\alpha N$
The procedure to implement this in python is
1. Create an empty vector to save the solution
   `sol = np.zeros(n)`
2. Set up the initial condition $N(t=0) = N_0$.
   `sol[0] = N0`
3. Create a time array to evaluate the sol and the time step
   `t_arr = np.linspace(0, tf, n)`, `dt = t_arr[i] - t_arr[0]`
4. Create the slope function
   ```
   def F(t, s):
       return -alpha*N
   ```
5. Implement the method by iterating over all n steps defined, a predictor step could be added.
   ```
   for i in range(n):
       sol[i+1] = sol[i] + dt/2 [F(t[i], sol[i]) + F(t[i+1], sol[i+1])]
   ```

2. **(2 points) SLURM**
   (a) Describe the role of SLURM in a high-performance computing (HPC) environment.

The slurm package is made for managing the resources in a HPC. So from this we can access to a specific computational resources to run a work as well as it provides information about arquitechture like partitions.

   (b) Provide an example of how resource requests are specified in a SLURM job script (e.g. when requesting a specific number of CPU cores and memory in a partition).

The header of our bash script should provide the information necessary to allocate our job.

```
#!/bash/bin
#sbatch --name: job
#sbatch --partition: partition_name
#sbatch --memory: 1G
#sbatch --time: hh:mm
#sbatch --cores: 5        (cpus-per-task)
#sbatch --output: job.out
#sbatch --error: job.err
"Here the script for running".
```

3. **(2 points) Secure Shell (SSH) protocol**

(a) Briefly explain what the SSH protocol is and how it works.

The SSH protocol is a asymmetric cryptographic protocol that maintain secure the connexion through internet between an user and a host. SSH protocol uses a private key and a public key to encrypt/decrypt information where the public key should be allocated in the HPC and the private key only in the user's device.

(b) Provide a syntax example on how to use it.

```
ssh -i ~/.ssh/key-name user-name@hpc-direction
```

4. **(2 points) Object-oriented programming (OOP)**

(a) List two key differences between the `@classmethod` and `@staticmethod` decorators.

1. @classmethod works with the class itself while @staticmethod does not "know" anything about the class.
2. @classmethod can access to the class with 'cls' while @staticmethod cannot, it only works with the input parameters that it could have.

(b) Explain the concept of encapsulation in object-oriented programming.

This concept is developed to protect or avoid accidental changes in attributes of a class. In this way, the user cannot access/change this protected objects. We learn about three protection levels in encapsulation (in Python), which are: self.var1 = var1 (not protected), self._var2 = var2 (protected) and self.__var3 = var3 (strong protection).

-0.5

5. **(2 points) Python parallelisation**

(a) Sequentially applying an edge-detection filter to high-resolution medical images to high-light anatomical structures takes a long time. Explain how the Python's `multiprocessing` module can help improve the performance of this image filtering task.

private

As we know images are multi-dimensional arrays (with 1, 2, or 3 layers). So we can distribute "portions" of the image to work on them simultaniously along a determined number of cores. Using the multiprocessing module the easiest way is using 'pool' to create an object for creating the processeses allocations, and then use the function of multiprocessing to distribute the work. Ok, but how are you detecting edges? Reconstruction?

(b) Describe the key steps involved in implementing a parallelised algorithm that uses `multiprocessing` to apply the edge-detection filter to these images.

1. Create a function that could be avaible to work on a portion of the image. available?
   I think that there are many ways for distributing the work but the more appropiate could be working on layers(3), so each processor would work simultaniously on each of them.

```
def edge_detector(i):
```

2. Creating the pool object. → `obj = Pool(processes = n_cpu)` (...)
3. Allocating the work on the corestor parallelizing. I don't remember the function name but it should be something like this: how? → map

```
result = obj.function(edge_detector, range(3)).
```

Aggregation?