Code Documentation

Charges.py

The charges.py file defines an abstract base class Charge and several concrete subclasses that represent different types of moving electric charges. These classes are designed for use in a MovingChargesField class.

Imports and Constants

numpy (np): Used for numerical calculations.
abc (ABC, abstractmethod): Used for creating an abstract base class and abstract methods.
scipy.constants: Provides physical constants like epsilon_0 (Permittivity of Free Space), pi, e (elementary charge), c (speed of light), and mu_0 (Permeability of Free Space).
Constants defined: eps, pi, e, c, u_0.

Subclasses

LinearAcceleratingCharge: Represents a charge with linearly increasing velocity.
LinearDeceleratingCharge: Represents a charge with linearly decreasing velocity.
LinearVelocityCharge: Represents a charge moving with a constant linear velocity.
OrbittingCharge: Represents a charge moving in an orbital path.
OscillatingCharge: Represents a charge moving back and forth in an oscillatory manner.

Each of these subclasses implements the abstract methods of Charge to provide the specific motion characteristics (position, velocity, acceleration) for the type of charge they represent. This involves defining the motion equations for each type of charge.

Class `Charge` (Abstract Base Class)

This is the base class for different types of charge motion.

Methods:

`__init__(self, pos_charge=True)`
- Initializes the charge.
- `pos_charge`: Boolean indicating if the charge is positive.

Abstract Methods: `xpos(t)`, `ypos(t)`, `zpos(t)`, `xvel(t)`, `yvel(t)`, `zvel(t)`, `xacc(t)`, `yacc(t)`, `zacc(t)`
- These methods, to be implemented by subclasses, determine the position, velocity, and acceleration in the x, y, and z dimensions at time `t`.

`retarded_time(self, tr, t, X, Y, Z)`

    - Returns the equation to solve for retarded time, based on Griffiths Eq. 10.55. This is used for electromagnetic calculations where the effects of changes in the charge's motion are felt after retarded time.

Class `OscillatingCharge` (Subclass of `Charge`)

The `OscillatingCharge` class is a subclass of `Charge` that models a charge undergoing oscillatory motion. Here is a detailed breakdown:

Constructor: `__init__`

- Initializes the charge with specific properties:
    - `pos_charge`: Boolean indicating if the charge is positive.
    - `start_position`: The initial position of the charge in 3D space. Preferably in nanometers.
    - `direction`: The direction of oscillation, normalized to unit length.
    - `amplitude`: The maximum displacement from the start position. Preferably in nanometers.
    - `max_speed`: The maximum speed of oscillation, used to calculate angular frequency.
    - `start_zero`: If `True`, the charge starts from rest.
    - `stop_t`: The time at which oscillation stops, if applicable.

Position Methods: `xpos`, `ypos`, `zpos`

- These methods calculate the position of the charge in x, y, and z dimensions, respectively, as a function of time `t`.
- The position is given by an oscillatory function (1 - cos(wt)) modulated by the amplitude and direction.
- If `start_zero` is `True`, the position remains at the start position for `t < 0`.
- If `stop_t` is specified, the position is fixed at the location at `stop_t` for `t > stop_t`.

Velocity Methods: `xvel`, `yvel`, `zvel`

- Calculate the velocity in x, y, and z dimensions.
- The velocity is the derivative of the position function, resulting in a sin(wt) function modulated by the amplitude and angular frequency.
- The velocity is zero before the start (if `start_zero` is `True`) and after the stop time (`stop_t`).

Acceleration Methods: `xacc`, `yacc`, `zacc`

- Calculate the acceleration in x, y, and z dimensions.
- The acceleration is given by the second derivative of the position function, resulting in a cos(wt) function modulated by the amplitude and angular frequency squared.
- The acceleration is zero before the start and after the stop time, similar to the velocity.

Additional Method: `get_period`

- Calculates the period of oscillation using the formula `2*pi/w`, where `w` is the angular frequency.

This class models a charge that oscillates along a certain direction, with the ability to specify start and stop times, amplitude of oscillation, and the direction of oscillation. It's useful for simulating physical phenomena where charges exhibit oscillatory motion, such as in certain electromagnetic fields or mechanical systems. The mathematical representation used in this class aligns with standard equations for simple harmonic motion in physics.

Class `OrbittingCharge` (Subclass of `Charge`)

The `OrbittingCharge` class, a subclass of `Charge`, represents a charge moving in an orbital motion. Here's a detailed explanation:

Constructor: `__init__`

- Initializes the charge with specific properties:
    - `pos_charge`: Indicates if the charge is positive.
    - `phase`: The initial phase angle in the orbital motion.
    - `amplitude`: Represents the radius of the orbital path.
    - `max_speed`: Maximum speed of the charge, used to calculate angular frequency.
    - `start_zero`: If `True`, the charge starts from rest.

Position Methods: `xpos`, `ypos`, `zpos`

- `xpos(t)`: Calculates the x-coordinate of the charge in its orbit at time `t` using the cosine function, modulated by the amplitude and angular frequency. Incorporates the initial phase and `start_zero` condition.

- `ypos(t)`: Similar to `xpos`, but uses the sine function for the y-coordinate.
- `zpos(t)`: Returns 0, indicating the motion is confined to the xy-plane.

Velocity Methods: `xvel`, `yvel`, `zvel`

- `xvel(t)`: Calculates the x-component of velocity, derived from the derivative of `xpos`, resulting in a sine function with a negative sign.
- `yvel(t)`: Calculates the y-component of velocity, derived from `ypos`, resulting in a cosine function.
- `zvel(t)`: Returns 0, as there's no motion in the z-direction.

Acceleration Methods: `xacc`, `yacc`, `zacc`

- `xacc(t)`: Computes the x-component of acceleration, derived from `xvel`, resulting in a cosine function with a negative sign.
- `yacc(t)`: Computes the y-component of acceleration, derived from `yvel`, resulting in a sine function with a negative sign.
- `zacc(t)`: Returns 0, as there's no acceleration in the z-direction.

Additional Method: `get_period`

- Calculates the period of the orbital motion using `2*np.pi/self.w`.

This class models a charge moving in a circular orbit within the xy-plane, providing functionality to specify the orbit's radius (amplitude), the speed, and an initial phase. The `start_zero` option allows the motion to start from a standstill. The mathematical representation used aligns with the equations for circular motion in physics, and it's useful for simulating scenarios like electrons in circular orbits or charged particles in magnetic fields.

Class `OscillatingOrbittingCharge` (Subclass of `Charge`)

The `OscillatingOrbittingCharge` class is an intricate subclass of `Charge`, combining features of oscillation and orbital motion. Here's a detailed breakdown:

Constructor: `__init__`

- Initializes the charge with properties relevant to both oscillatory and orbital motion:
  - `pos_charge`: Indicates if the charge is positive.
  - `phase`: The initial phase angle.

- `radius`: The radius of the orbital path.
- `w`: Angular frequency of orbital motion.
- `max_speed`: Maximum speed, used to calculate the amplitude of oscillation.
- `A`: Amplitude of the oscillation, calculated as `max_speed/(radius*w)`.
- `start_zero`: Determines if the charge starts from rest.

Position Methods: `xpos`, `ypos`, `zpos`

- `xpos(t)` and `ypos(t)`: Calculate the x and y positions of the charge, combining oscillatory and orbital motions. The positions are given by sinusoidal functions, with the amplitude of oscillation (`A`) modulated by another sinusoidal function representing the orbital motion. The `phase` is also considered.
- `zpos(t)`: Returns 0, indicating motion is confined to the xy-plane.

Velocity Methods: `xvel`, `yvel`, `zvel`

- `xvel(t)` and `yvel(t)`: Calculate the x and y components of velocity, derived from the position functions. These involve complex combinations of sinusoidal functions representing both oscillatory and orbital velocities.
- `zvel(t)`: Returns 0, as there's no motion in the z-direction.

Acceleration Methods: `xacc`, `yacc`, `zacc`

- `xacc(t)` and `yacc(t)`: Compute the acceleration in the x and y directions. The acceleration components are derived from the velocity functions and involve even more complex combinations of trigonometric functions.
- `zacc(t)`: Returns 0, as there's no acceleration in the z-direction.

Additional Method: `get_period`

- Calculates the period of the oscillatory motion using `2*np.pi/self.w`.

This class represents a sophisticated model of a charge that exhibits both oscillatory and orbital motions in the xy-plane. The oscillatory motion is superimposed on the orbital motion, creating a complex path. Such a class could be used to simulate charges in complex electromagnetic fields or in scenarios where multiple forces influence the charge's motion. The implementation requires a deep understanding of both trigonometric functions and their derivatives, as it involves layered oscillatory behaviors.

Class `LinearAcceleratingCharge` (Subclass of `Charge`)

The `LinearAcceleratingCharge` class represents a point charge that undergoes linear acceleration, specifically in the x-direction. Here's a detailed explanation:

Constructor: `__init__`

- Initializes the charge with properties related to linear acceleration:
    - `pos_charge`: Indicates if the charge is positive.
    - `acceleration`: The constant acceleration of the charge in the x-direction.
    - `stop_t`: The time at which acceleration stops. If not provided, it's calculated to ensure the velocity does not exceed the speed of light (`c`).

Position Method: `xpos`

- Calculates the x-position of the charge as a function of time `t`.
- Implements the standard kinematic equation for position under constant acceleration: `0.5 * acceleration * t**2`.
- Ensures the position remains zero for `t < 0` and calculates position correctly for `t > stop_t`, taking into account the constant velocity after stopping acceleration.

Velocity Method: `xvel`

- Calculates the x-velocity of the charge.
- Follows the linear kinematic equation for velocity under constant acceleration: `acceleration * t`.
- Velocity is zero for `t < 0` and is capped at `acceleration * stop_t` for `t > stop_t`.

Acceleration Method: `xacc`

- Provides a constant acceleration value (`acceleration`) for `t` between 0 and `stop_t`.
- Acceleration is zero for `t < 0` and `t > stop_t`.

Position, Velocity, and Acceleration Methods for y and z Directions (`ypos`, `zpos`, `yvel`, `zvel`, `yacc`, `zacc`)

- All these methods return 0, indicating no motion or acceleration in the y and z directions.

This class is designed to model a charge that accelerates linearly from rest and then continues at a constant velocity after a specified time (`stop_t`). The implementation adheres to the basic principles of kinematics in one dimension, with special consideration to the relativistic constraint that the velocity should not exceed the speed of light. This class could be useful in simulations involving particle accelerators or other scenarios where charges experience linear acceleration.

Class `LinearDeceleratingCharge` (Subclass of `Charge`)

This class represents a point charge that decelerates in the x-direction from an initial speed.

Constructor: `__init__`

- Initializes the charge with specific deceleration properties:
  - `pos_charge`: Indicates if the charge is positive.
  - `deceleration`: The constant deceleration in the x-direction.
  - `initial_speed`: The initial velocity of the charge.
  - `stop_t`: The time at which deceleration stops, calculated so the velocity becomes zero.

Position Method: `xpos`

- Calculates the x-position under linear deceleration, using kinematic equations.
- For `t > 0`, it accounts for the initial speed and deceleration.
- For `t > stop_t`, the position remains constant, indicating the charge has stopped.

Velocity Method: `xvel`

- Calculates the x-velocity, decreasing linearly with time.
- Velocity is capped at zero once `t > stop_t`.

Acceleration Method: `xacc`

- Provides a constant negative acceleration (`-deceleration`) for `t` between 0 and `stop_t`.
- Acceleration is zero for `t < 0` and `t > stop_t`.

Methods for y and z Directions

- All these methods (`ypos`, `zpos`, `yvel`, `zvel`, `yacc`, `zacc`) return 0, indicating no motion or acceleration in the y and z directions.

`LinearDeceleratingCharge` models a charge slowing down linearly.

Class `LinearVelocityCharge` (Subclass of `Charge`)

This class models a point charge moving at a constant velocity in the x-direction.

Constructor: `__init__`

- Initializes the charge with specific velocity properties:
  - `pos_charge`: Indicates if the charge is positive.
  - `speed`: The constant speed in the x-direction.
  - `init_pos`: The initial position of the charge.

Position Method: `xpos`

- Calculates the x-position using the formula `speed * t + init_pos`.

Velocity Method: `xvel`

- Returns the constant velocity (`speed`).

Acceleration Method: `xacc`

- Returns 0, indicating no acceleration in the x-direction.

Methods for y and z Directions

- All these methods (`ypos`, `zpos`, `yvel`, `zvel`, `yacc`, `zacc`) return 0, indicating no motion or acceleration in the y and z directions.

`LinearVelocityCharge` represents a charge moving at a constant speed, which can be useful in various physical simulations or theoretical studies of charge motion.

Class `LinearVelocityCharge` (Subclass of `Charge`)

The `LinearAcceleratingDeceleratingCharge` class in Python represents a charge that undergoes a sequence of motions: accelerating, moving at constant velocity, decelerating, and finally stopping. This class is a subclass of `Charge` and it models a more complex motion compared to simple linear acceleration or deceleration. Let's analyze it in detail:

Constructor: `__init__(self, pos_charge, acceleration, deceleration, stop_t, acc_impulse_t, dec_impulse_t)`

- Purpose: Sets up the charge with initial conditions for its motion.
- Parameters:
    - `pos_charge`: Boolean indicating if the charge is positive.
    - `acceleration`: Acceleration value for the initial phase.
    - `deceleration`: Deceleration value for the final phase.
    - `stop_t`: The time at which the charge completely stops.
    - `acc_impulse_t`: The time at which acceleration phase ends and constant velocity begins.
    - `dec_impulse_t`: The time at which deceleration phase begins.

Motion Phases

Accelerating Phase: From start until `acc_impulse_t`.
Constant Velocity Phase: From `acc_impulse_t` until `dec_impulse_t`.
Decelerating Phase: From `dec_impulse_t` until `stop_t`.
Stopped: After `stop_t`.

Methods

Position (`xpos`):

- Functionality: Calculates the x-position of the charge at any given time `t`.
- Uses kinematic equations to compute the position for each phase of motion:
    - During acceleration: `x = 0.5 * acceleration * t^2`.
    - During constant velocity: Distance traveled is computed based on the velocity at the end of the acceleration phase.
    - During deceleration: The position is calculated considering the deceleration, starting from the end of the constant velocity phase.

Velocity (`xvel`):

- Functionality: Computes the x-velocity of the charge at any time `t`.
- During acceleration: `v = acceleration * t`.
- After `acc_impulse_t` and before `dec_impulse_t`: Velocity is constant.
- During deceleration: Velocity decreases linearly with time.
- Stops completely after `stop_t`.

Acceleration (`xacc`):

- Functionality: Determines the x-acceleration of the charge at any time `t`.
- Constant `acceleration` until `acc_impulse_t`.
- Zero during constant velocity phase.
- Constant `-deceleration` during deceleration phase.
- Zero after `stop_t`.

Y and Z Directions (`ypos`, `zpos`, `yvel`, `zvel`, `yacc`, `zacc`):

- All these methods return 0, indicating no motion or acceleration in the y and z directions.

Use Case

This class can be used in physics simulations where charges undergo complex motion, such as in certain types of particle accelerators or in electromagnetic field studies. It provides a more realistic model of a charge's motion that includes distinct phases of acceleration, constant speed, and deceleration, which is common in many physical systems.

MovingChargesField.py

The `MovingChargesField` class in `field_calculations.py` provides a comprehensive framework for computing electromagnetic fields and potentials resulting from the movement of point charges. Let's delve into a more detailed analysis:

Overview

The class uses the Liénard-Wiechert potentials, a fundamental solution to Maxwell's equations, applicable in scenarios involving moving charges. These potentials consider the finite speed of light and the consequent time delays (retarded time) in the influence of charge movements.

Methods Breakdown

Constructor: `__init__(self, charges, h=1e-20)`

- Purpose: Initializes the `MovingChargesField` object.
- Parameters:
    - `charges`: A list of charge objects (instances of `Charge` class or its subclasses) whose fields are to be calculated.
    - `h`: Tolerance level for numerical optimization in calculating the retarded time, set to a very small number to achieve high precision.
- Functionality:
    - Converts a single charge object to a list if necessary, ensuring compatibility with multiple or single charge scenarios.

`calculate_E(self, t, X, Y, Z, pcharge_field='Total', plane=False)`

- Purpose: Calculates the electric field (`E`) at given spatial coordinates and time.
- Parameters:
    - `t`: Time at which the field is calculated.
    - `X`, `Y`, `Z`: 3D meshgrid arrays representing the spatial points where the electric field is to be calculated.
    - `pcharge_field`: Option to calculate fields due to 'Velocity', 'Acceleration', or both ('Total').
    - `plane`: If `True`, returns 2D field arrays, else 3D arrays.
- Functionality:

    - Iterates over each charge to compute its contribution to the field at each point.
    - Uses the `optimize.newton` method to find the retarded time (`tr`) for each charge, which is crucial to account for the time delay in electromagnetic effects.
    - Sums up the contributions from all charges to get the total field.
    - Uses `_calculate_individual_E` method for calculating the field from each charge.

`calculate_B(self, t, X, Y, Z, pcharge_field='Total', plane=False)`

- Purpose: Computes the magnetic field (`B`).
- Functionality:
    - Similar to `calculate_E`, but focuses on the magnetic aspect.
    - Uses the calculated electric field components and spatial positions to compute the magnetic field via the cross-product formula, as per Griffiths Eq. 10.73.

`_calculate_individual_E(self, charge, tr, X, Y, Z, pcharge_field)`

- Purpose: Helper function to calculate the electric field contribution from an individual charge.
- Functionality:
    - Utilizes advanced electromagnetism formulas (Griffiths Eqs. 10.71, 10.72) to compute the field, factoring in the charge's retarded position, velocity, and acceleration.

`calculate_potentials(self, t, X, Y, Z, plane=False)`

- Purpose: Calculates the scalar and vector potentials (`V`, `Ax`, `Ay`, `Az`).
- Functionality:
    - Employs the Liénard-Wiechert potential formula (Griffiths Eq. 10.53) for each charge, again considering the retarded time.
    - Computes potentials at each point, adding the individual contributions from each charge.

`calculate_Poynting(self, t, X, Y, Z, plane=False)`

- Purpose: Computes the Poynting vector (`S`), which represents the power per unit area carried by an electromagnetic wave.
- Functionality:
    - Uses the calculated electric field intensity to compute the Poynting vector based on Griffiths Eq. 11.67.
    - Suitable for analyzing energy transmission in electromagnetic fields.

Use Cases and Significance

This class is highly relevant in advanced physics and engineering fields, particularly in electromagnetism and optics. It allows for the numerical simulation of complex scenarios involving moving charges, such as radiation from antennas, behavior of particles in accelerators, and electromagnetic wave propagation. The inclusion of retarded time makes it suitable for analyzing dynamic systems where the time-dependent nature of electromagnetic interactions is critical. This class can serve as a powerful tool in both academic research and practical engineering applications, offering insights into the behavior of electromagnetic fields and potentials in real-time scenarios involving moving charges.