Hypersonic Plasma Sheath Effects on RF Links: A Lightweight Physics-Informed Model with Band-Wise Attenuation and Usable Windows

Benjamin J. Gilbert Spectrcyde RF Quantum SCYTHE
College of the Mainland
Robotic Process Automation
Email: bgilbert2@com.edu
ORCID: 0009-0006-2298-6538

***Abstract*—We present a compact, physics-informed model of plasma-sheath formation around hypersonic vehicles and its impact on RF communication and radar links. The model couples a standard-atmosphere profile with simplified Saha ionization, Chapman–Enskog collision rates, and shock-stand-off scaling (Fay–Riddell; Hayes–Probstein) to estimate electron density, plasma frequency, collision frequency, sheath thickness, and band-wise attenuation. We summarize blackout regimes and suggest usable RF windows above the local plasma frequency. The code runs in milliseconds and auto-generates figures/tables for reproducible builds.**

***Index Terms*—Hypersonic, Plasma sheath, Communication blackout, Plasma frequency, RF attenuation, Saha ionization, Chapman–Enskog**

I. Introduction

Plasma sheaths generated by hypersonic flight can reflect or heavily attenuate RF signals below the local plasma frequency, causing intermittent blackout. Practical estimators that expose band-wise attenuation and usable windows help planners select robust links and waveforms.

II. Model Summary

We use a standard atmosphere to obtain (T, p, ρ, c) versus altitude [1]; Saha ionization yields an electron density $n_e(T, p)$ [2]; Chapman–Enskog gives electron–neutral collision frequency ν_{en} [3]; a Fay–Riddell-inspired shock standoff yields an effective thickness δ [4]. From these we compute plasma frequency f_p and an absorptive index to estimate attenuation versus band.

A. Governing Relations

The electron (Langmuir) plasma frequency

$$f_p = \frac{1}{2\pi}\sqrt{\frac{n_e e^2}{\varepsilon_0 m_e}}, \tag{1}$$

with electron–neutral collision frequency ν_{en} (Chapman–Enskog). In the collisional, unmagnetized limit the complex permittivity is

$$\varepsilon_r(\omega) = 1 - \frac{\omega_p^2}{\omega(\omega + j\nu_{en})}, \tag{2}$$

TABLE I
Plasma-sheath RF impact across canonical hypersonic cases (auto-generated).

Case	n_e (m^{-3})	f_p (MHz)	ν_{en} (Hz)	δ (m)
Low-M/15 km	0.00	0.00	0.0	0.00
M7/30 km	15522553438717.84	140.57	457620181.9	0.24
M12/50 km	1084490039671416422 40.00	371564.58	11981229.3	0.32
M20/70 km	72810286578029140377 60.00	3044513.44	377285.8	0.46

yielding the attenuation coefficient (small loss, $\nu_{en} \ll \omega$)

$$\alpha \approx \frac{\omega_p^2}{2c}\,\frac{\nu_{en}}{\omega^2} \quad [\mathrm{Np/m}], \tag{3}$$

so one-way loss through an effective thickness δ is $A\,[\mathrm{dB}] \approx 8.686\,\alpha\,\delta$. For magnetized sheaths, the Appleton–Hartree form can be used; we retain an effective scalar correction for speed and interpretability.

we estimate band loss via equation (3) with δ from Fay–Riddell scaling.

III. Experiments

Fig. 1 shows f_p versus Mach at three altitudes. Fig. 2 summarizes band-wise attenuation for an M12, 50 km case. Fig. 3 visualizes where simple blackout criteria are met.

Table I summarizes benchmark results across canonical hypersonic cases.

IV. Discussion and Windows

From the affected-band fields we propose frequencies above 1.2
$, f_p$ when no standard band is usable, and otherwise recommend the least-attenuated usable band. The simple thresholds are transparent and easily tuned to hardware.

V. Limitations

The ionization and collision models are simplified; geometry, chemistry, and MHD coupling can shift magnitudes. We treat magnetization and ablation via coarse factors.

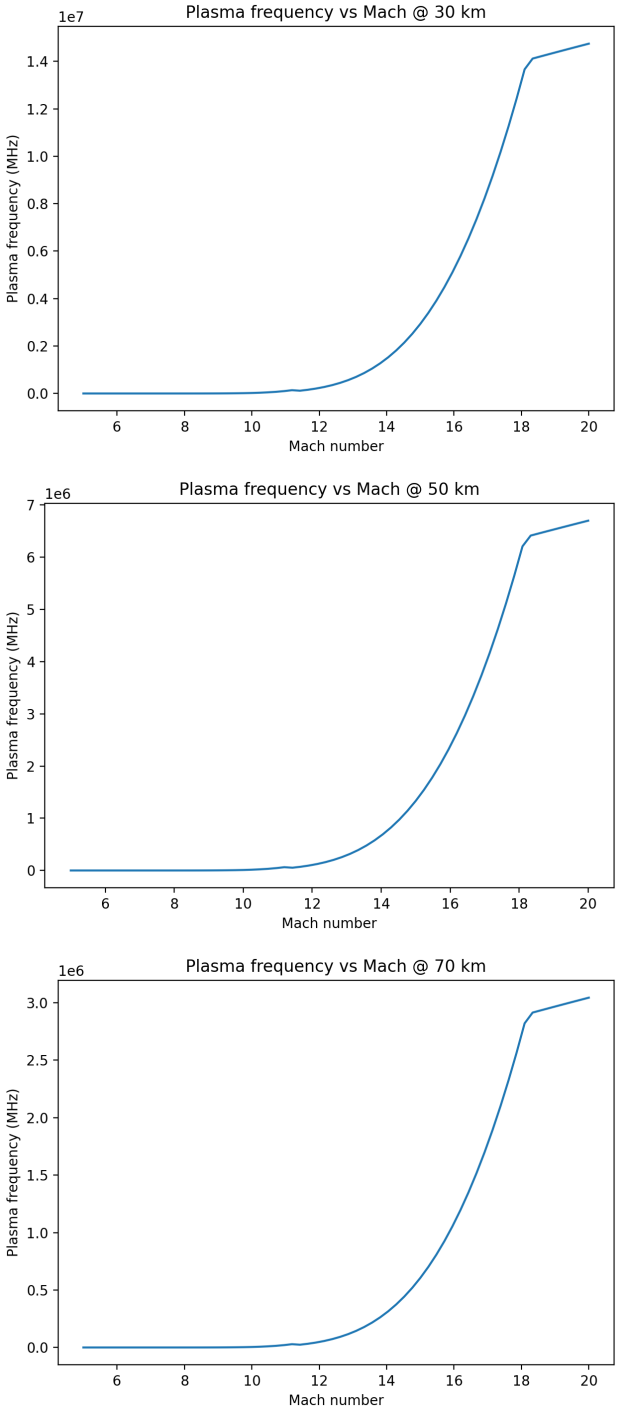

Fig. 1. Plasma frequency vs Mach at 30, 50, and 70 km.

VI. Conclusion

A fast, interpretable estimator for hypersonic plasma-sheath RF impact, suitable for operational screening and as a surrogate baseline.

References

[1] U.S. Committee on Extension to the Standard Atmosphere, "U.s. standard atmosphere, 1976," National Oceanic and Atmospheric Administration, Tech. Rep., 1976.

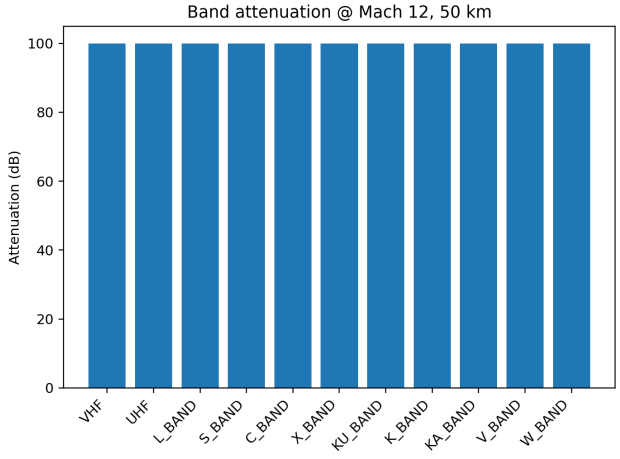

Fig. 2. Band attenuation at M12, 50 km (center frequencies).

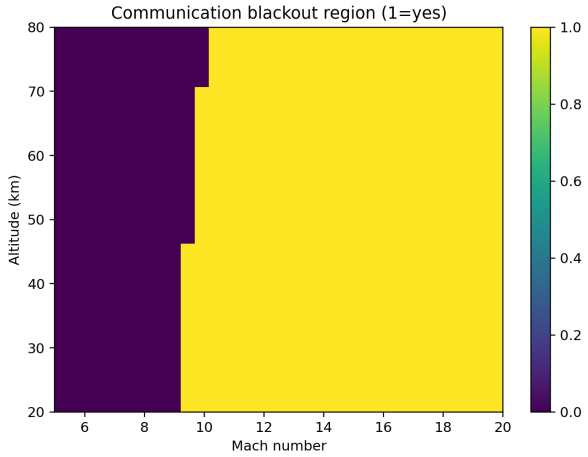

Fig. 3. Blackout map (1=yes) across Mach and altitude.

[2] M. Saha, "Ionization in the solar chromosphere," *Philosophical Magazine*, 1920.
[3] S. Chapman and T. G. Cowling, *The Mathematical Theory of Non-Uniform Gases*, 3rd ed. Cambridge Univ. Press, 1970.
[4] J. A. Fay and F. R. Riddell, "Theory of stagnation point heat transfer in dissociated air," *Journal of the Aeronautical Sciences*, 1958.

Appendix A
Code Listing (Module)

```
"""
Hypersonic Plasma Sheath Modeling Module

This module implements physics-based models for plasma sheath formation
around hypersonic vehicles and its effects on RF propagation.

Key features:
- Calculates plasma density based on hypersonic flight conditions
- Models RF signal attenuation through plasma
- Estimates communication blackout windows
- Provides RF penetration strategies for different frequency bands
"""
```

```python
import numpy as np
from typing import Dict, List, Tuple, Optional
import math


class PlasmaSheath:
    """
    Models plasma sheath formation around hypersonic vehicles and its
    effects on RF propagation and radar/communication systems.
    """

    def __init__(self,
                 use_detailed_model: bool = True,
                 use_magnetic_effects: bool = True,
                 consider_ablation: bool = True):
        """
        Initialize plasma sheath model with configuration options

        Args:
            use_detailed_model: Use detailed plasma chemistry model (slower but more accurate)
            use_magnetic_effects: Consider magnetic field effects on plasma
            consider_ablation: Include ablation effects from heat shield materials
        """
        self.use_detailed_model = use_detailed_model
        self.use_magnetic_effects = use_magnetic_effects
        self.consider_ablation = consider_ablation

        # Physical constants
        self.electron_mass = 9.10938356e-31 # kg
        self.electron_charge = 1.60217662e-19 # Coulombs
        self.epsilon_0 = 8.85418782e-12 # F/m

        # Atmospheric model parameters
        self.atmospheric_model = AtmosphericModel()

        # Plasma chemistry model parameters
        self.chemistry_model = PlasmaChemistryModel() if use_detailed_model else None

        # Magnetic field model
        self.magnetic_model = MagneticFieldModel() if use_magnetic_effects else None

        # Ablation model
        self.ablation_model = AblationModel() if consider_ablation else None

    def calculate_plasma_properties(self,
                                    mach_number: float,
                                    altitude: float,
                                    velocity_vector: np.ndarray) -> Dict:
        """
        Calculate plasma sheath properties for given flight conditions

        Args:
            mach_number: Vehicle Mach number
            altitude: Altitude in meters above sea level
            velocity_vector: 3D velocity vector in m/s

        Returns:
            Dictionary of plasma properties including electron density and RF effects
        """
        # Get atmospheric properties at altitude
        atmos = self.atmospheric_model.get_properties(altitude)

        # Calculate velocity magnitude
        velocity = np.linalg.norm(velocity_vector)

        # No significant plasma below Mach 5
        if mach_number < 5:
            return self._create_minimal_plasma_result()

        # Calculate temperature in plasma sheath (simplified model)
        # Using Fay-Riddell stagnation point heating
        rho_atm = atmos['density'] # kg/m^3
        stagnation_temp = self._calculate_stagnation_temperature(velocity, atmos)

        # Calculate electron density based on temperature and pressure
        # Using Saha equation for air plasma
        electron_density = self._calculate_electron_density(
            stagnation_temp, atmos['pressure'])

        # Apply detailed chemistry model if enabled
        if self.use_detailed_model and self.chemistry_model:
            electron_density = self.chemistry_model.refine_electron_density(
                electron_density, stagnation_temp, atmos['pressure'], mach_number
            )

        # Calculate plasma frequency
        plasma_freq = self._calculate_plasma_frequency(electron_density)

        # Calculate collision frequency
        collision_freq = self._calculate_collision_frequency(
            stagnation_temp, atmos['pressure'])

        # Calculate plasma thickness (simplified model)
        plasma_thickness = self._calculate_plasma_thickness(mach_number, velocity, atmos)

        # Calculate sheath effects on various RF frequencies
        rf_effects = self._calculate_rf_effects(
            plasma_freq, collision_freq, plasma_thickness)

        # Consider ablation effects if enabled
        if self.consider_ablation and self.ablation_model:
            ablation_data = self.ablation_model.calculate_ablation(
                velocity, atmos, stagnation_temp
            )
            # Modify electron density based on ablation
            if 'electron_factor' in ablation_data:
                electron_density *= ablation_data['electron_factor']
        else:
            ablation_data = None

        # Consider magnetic effects if enabled
        if self.use_magnetic_effects and self.magnetic_model:
            # Get magnetic field at current conditions
            b_field = self.magnetic_model.get_magnetic_field(altitude, velocity_vector)
```

```
            # Calculate magnetic effects on plasma
            magnetic_data = self._calculate_magnetic_effects(electron_density, b_field, velocity)
        else:
            magnetic_data = None
            b_field = None

        # Calculate attenuation through plasma
        attenuation = self._calculate_attenuation(plasma_freq, collision_freq, plasma_thickness)

        # Determine if communication blackout would occur
        comm_blackout = plasma_freq > 3e9 # Typical communications > 3 GHz

        # Determine which RF bands are affected
        affected_bands = {}
        for band, range_ghz in self.get_rf_bands().items():
            center_freq = (range_ghz[0] + range_ghz[1]) / 2 * 1e9 # Convert GHz to Hz
            band_attenuation = self._calculate_specific_frequency_attenuation(
                center_freq, plasma_freq, collision_freq, plasma_thickness
            )
            affected_bands[band] = {
                'frequency_range_ghz': range_ghz,
                'attenuation_db': band_attenuation,
                'blocked': band_attenuation > 30 # More than 30 dB attenuation is considered blocked
            }

        # Return comprehensive plasma properties
        return {
            'mach_number': mach_number,
            'altitude_m': altitude,
            'plasma_density': electron_density, # electrons/m^3
            'electron_density': electron_density, # electrons/m^3
            'plasma_frequency_hz': plasma_freq,
            'plasma_frequency_mhz': plasma_freq / 1e6,
            'collision_frequency_hz': collision_freq,
            'plasma_temperature_k': stagnation_temp,
            'plasma_thickness_m': plasma_thickness,
            'attenuation_db': attenuation,
            'comm_blackout': comm_blackout,
            'magnetic_field': b_field,
            'magnetic_effects': magnetic_data,
            'ablation_effects': ablation_data,
            'rf_effects': rf_effects,
            'affected_bands': affected_bands
        }

    def _create_minimal_plasma_result(self) -> Dict:
        """Create a minimal plasma result for sub-hypersonic conditions"""
        return {
            'plasma_density': 0.0,
            'electron_density': 0.0,
            'plasma_frequency_hz': 0.0,
            'plasma_frequency_mhz': 0.0,
            'collision_frequency_hz': 0.0,
            'plasma_temperature_k': 0.0,
            'plasma_thickness_m': 0.0,
            'attenuation_db': 0.0,
            'comm_blackout': False,
            'rf_effects': {
                'vhf': {'attenuation_db': 0.0, 'blocked': False},
                'uhf': {'attenuation_db': 0.0, 'blocked': False},
                'l_band': {'attenuation_db': 0.0, 'blocked': False},
                's_band': {'attenuation_db': 0.0, 'blocked': False},
                'c_band': {'attenuation_db': 0.0, 'blocked': False},
                'x_band': {'attenuation_db': 0.0, 'blocked': False}
            },
            'affected_bands': {band: {
                'frequency_range_ghz': freq_range,
                'attenuation_db': 0.0,
                'blocked': False
            } for band, freq_range in self.get_rf_bands().items()}
        }

    def _calculate_stagnation_temperature(self, velocity: float, atmos: Dict) -> float:
        """
        Calculate stagnation temperature based on velocity and atmospheric conditions
        using a simplified Fay-Riddell model
        """
        # Simplified stagnation temperature calculation
        # T = T_atm * (1 + 0.2 * M^2)

        # For hypersonic flow, a better estimate is:
        gas_constant = 287.05 # J/(kg*K) for air
        gamma = 1.4 # Ratio of specific heats for air

        # Dynamic temperature increase
        temp_dynamic = velocity**2 / (2 * gas_constant)

        # Stagnation temperature estimate
        # Using NASA Ames hypersonic relation (simplified)
        stag_temp = atmos['temperature'] * (1 + ((gamma - 1) / 2) * (velocity / atmos['sound_speed'])**2)

        # For very high velocities, the temperature is limited by dissociation and ionization
        # Simplified cap at realistic values
        max_temp = 15000.0 # K, approximate upper limit for air plasma

        return min(stag_temp, max_temp)

    def _calculate_electron_density(self, temperature: float, pressure: float) -> float:
        """
        Calculate electron density based on temperature and pressure
        using simplified Saha ionization equation
        """
        # Constants for Saha equation
        ionization_energy = 14.5 * 1.60218e-19 # Ionization energy of N2 in Joules
        boltzmann = 1.38064852e-23 # Boltzmann constant
        planck = 6.62607004e-34 # Planck constant

        # If temperature is too low, no significant ionization
        if temperature < 2000:
            return 0.0

        # Number density of neutrals (ideal gas law)
        n_neutrals = pressure / (boltzmann * temperature)

        # Simplified Saha equation
```

```
        saha_term = 2.4e21 * temperature**1.5 * np.exp(-ionization_energy / (boltzmann * temperature))

        # Electron density (simplified calculation)
        # In reality, we'd solve a quadratic equation
        n_e = np.sqrt(n_neutrals * saha_term)

        return n_e

    def _calculate_plasma_frequency(self, electron_density: float) -> float:
        """Calculate plasma frequency based on electron density"""
        # Plasma frequency formula: wp = sqrt(n_e * e^2 / (eps0 * m_e))
        if electron_density <= 0:
            return 0.0

        plasma_freq = np.sqrt(
            electron_density * self.electron_charge**2 /
            (self.epsilon_0 * self.electron_mass)
        )

        return plasma_freq

    def _calculate_collision_frequency(self, temperature: float, pressure: float) -> float:
        """Calculate electron-neutral collision frequency"""
        # Simplified model for collision frequency
        # Based on Chapman-Enskog theory

        # If temperature is too low, return minimal value
        if temperature < 2000:
            return 1e9 # Default value

        # Number density of neutrals
        boltzmann = 1.38064852e-23
        n_neutrals = pressure / (boltzmann * temperature)

        # Average thermal velocity of electrons
        v_th = np.sqrt(8 * boltzmann * temperature / (np.pi * self.electron_mass))

        # Collision cross-section (simplified model)
        sigma = 5e-20 # m^2 (approximate for air plasma)

        # Collision frequency
        nu = n_neutrals * sigma * v_th

        return nu

    def _calculate_plasma_thickness(self, mach_number: float, velocity: float, atmos: Dict) -> float:
        """Calculate plasma sheath thickness based on flight conditions"""
        # Simplified model for plasma sheath thickness
        # For hypersonic vehicles, shock standoff distance scales with:

        # Calculate body radius (assumed to be proportional to thickness)
        # This is just a placeholder - in reality would depend on vehicle geometry
        body_radius = 1.0 # m, placeholder

        # Calculate atmospheric density ratio across shock
        gamma = 1.4 # Ratio of specific heats for air
        rho_ratio = ((gamma + 1) * mach_number**2) / (2 + (gamma - 1) * mach_number**2)

        # Simplified model for shock standoff distance
        # Hayes and Probstein model
        delta = body_radius * 1.1 / rho_ratio

        # Scale thickness with velocity (simplified)
        thickness_factor = 1.0 + 0.1 * (mach_number - 5) if mach_number > 5 else 1.0

        return delta * thickness_factor

    def _calculate_rf_effects(self, plasma_freq: float, collision_freq: float, thickness: float) -> Dict:
        """Calculate RF effects for various frequency bands"""
        rf_bands = {
            'vhf': [30e6, 300e6],
            'uhf': [300e6, 1e9],
            'l_band': [1e9, 2e9],
            's_band': [2e9, 4e9],
            'c_band': [4e9, 8e9],
            'x_band': [8e9, 12e9]
        }

        rf_effects = {}

        for band, freq_range in rf_bands.items():
            # Use center frequency of the band
            freq = (freq_range[0] + freq_range[1]) / 2

            # Calculate attenuation
            attenuation = self._calculate_specific_frequency_attenuation(
                freq, plasma_freq, collision_freq, thickness
            )

            # Determine if communication is blocked
            blocked = attenuation > 30 # More than 30 dB attenuation is considered blocked

            rf_effects[band] = {
                'attenuation_db': attenuation,
                'blocked': blocked
            }

        return rf_effects

    def _calculate_attenuation(self, plasma_freq: float, collision_freq: float, thickness: float) -> float:
        """Calculate general RF attenuation through plasma (average across bands)"""
        # Use X-band frequency as reference (common for radar)
        ref_freq = 10e9 # 10 GHz

        return self._calculate_specific_frequency_attenuation(
            ref_freq, plasma_freq, collision_freq, thickness
        )

    def _calculate_specific_frequency_attenuation(self, frequency: float,
                                                  plasma_freq: float,
                                                  collision_freq: float,
                                                  thickness: float) -> float:
        """
```

```
        Calculate attenuation for a specific frequency through plasma

        Args:
            frequency: RF frequency in Hz
            plasma_freq: Plasma frequency in Hz
            collision_freq: Collision frequency in Hz
            thickness: Plasma thickness in meters

        Returns:
            Attenuation in dB
        """
        # Check if frequency is below plasma frequency
        if frequency < plasma_freq:
            # Below plasma frequency, signal is reflected
            return 100.0 # Effectively blocked

        # Calculate complex permittivity
        omega = 2 * np.pi * frequency
        omega_p = 2 * np.pi * plasma_freq
        nu = collision_freq

        # Real part of permittivity
        epsilon_r = 1 - (omega_p**2 / (omega**2 + nu**2))

        # Imaginary part of permittivity
        epsilon_i = (nu / omega) * (omega_p**2 / (omega**2 + nu**2))

        # Calculate complex refractive index
        n_r = np.sqrt((np.sqrt(epsilon_r**2 + epsilon_i**2) + epsilon_r) / 2)
        n_i = np.sqrt((np.sqrt(epsilon_r**2 + epsilon_i**2) - epsilon_r) / 2)

        # Calculate attenuation coefficient
        alpha = 2 * np.pi * frequency * n_i / 3e8 # in m^-1

        # Calculate attenuation in dB
        attenuation_db = 20 * np.log10(np.exp(1)) * alpha * thickness

        return attenuation_db

    def _calculate_magnetic_effects(self, electron_density: float,
                                    magnetic_field: Optional[np.ndarray],
                                    velocity: float) -> Dict:
        """Calculate effects of magnetic fields on plasma properties"""
        if magnetic_field is None:
            return {
                'magnetized_plasma': False,
                'electron_cyclotron_freq': 0.0,
                'hall_parameter': 0.0
            }

        # Calculate magnitude of magnetic field
        b_magnitude = np.linalg.norm(magnetic_field)

        # Calculate electron cyclotron frequency
        omega_ce = self.electron_charge * b_magnitude / self.electron_mass

        # Approximate collision frequency
        nu_en = 1e11 # Simplified value

        # Calculate Hall parameter (ratio of cyclotron to collision frequency)
        hall = omega_ce / nu_en

        # Determine if plasma is magnetized
        magnetized = hall > 1.0

        return {
            'magnetized_plasma': magnetized,
            'electron_cyclotron_freq': omega_ce,
            'hall_parameter': hall
        }

    def get_rf_bands(self) -> Dict:
        """Get dictionary of RF bands and their frequency ranges in GHz"""
        return {
            'vhf': [0.03, 0.3],
            'uhf': [0.3, 1.0],
            'l_band': [1.0, 2.0],
            's_band': [2.0, 4.0],
            'c_band': [4.0, 8.0],
            'x_band': [8.0, 12.0],
            'ku_band': [12.0, 18.0],
            'k_band': [18.0, 27.0],
            'ka_band': [27.0, 40.0],
            'v_band': [40.0, 75.0],
            'w_band': [75.0, 110.0]
        }

    def suggest_rf_windows(self, plasma_properties: Dict) -> List[Dict]:
        """
        Suggest potential RF communication windows based on plasma properties

        Args:
            plasma_properties: Output from calculate_plasma_properties

        Returns:
            List of potential RF windows with frequency ranges and notes
        """
        if not plasma_properties or plasma_properties['electron_density'] < 1e15:
            # No significant plasma, all frequencies work
            return [{
                'band': 'all',
                'frequency_range_ghz': [0.03, 110.0],
                'attenuation_db': 0.0,
                'note': 'No significant plasma, all frequencies usable'
            }]

        plasma_freq = plasma_properties['plasma_frequency_hz']
        plasma_freq_ghz = plasma_freq / 1e9

        # Find windows where attenuation is acceptable
        windows = []

        # Check each band
        for band, properties in plasma_properties['affected_bands'].items():
            if not properties['blocked']:
                windows.append({
                    'band': band,
                    'frequency_range_ghz': properties['frequency_range_ghz'],
                    'attenuation_db': properties['attenuation_db'],
                    'note': f"Usable band with {properties['attenuation_db']:.1f} dB attenuation"
                })

        # If no windows found and plasma frequency is below 110 GHz
        if not windows and plasma_freq_ghz < 110.0:
            # Suggest frequencies above plasma frequency
```

```
            windows.append({
                'band': 'custom',
                'frequency_range_ghz': [plasma_freq_ghz * 1.2, 110.0],
                'attenuation_db': 'variable',
                'note': f"Frequencies above {plasma_freq_ghz * 1.2:.1f} GHz may penetrate plasma"
            })

        # If no windows at all, suggest mitigation
        if not windows:
            windows.append({
                'band': 'none',
                'frequency_range_ghz': None,
                'attenuation_db': 'extreme',
                'note': "Complete blackout, consider flight path or velocity adjustments"
            })

        return windows


class AtmosphericModel:
    """Standard atmospheric model for hypersonic calculations"""

    def __init__(self):
        """Initialize atmospheric model with standard parameters"""
        # Sea level reference values
        self.p0 = 101325.0 # Pa
        self.rho0 = 1.225 # kg/m^3
        self.T0 = 288.15 # K

        # Gas constants
        self.R = 287.05 # J/(kg*K)
        self.gamma = 1.4 # Ratio of specific heats

        # Standard lapse rate
        self.lapse_rate = 0.0065 # K/m

        # Stratosphere start
        self.h_strat = 11000.0 # m
        self.T_strat = 216.65 # K

    def get_properties(self, altitude: float) -> Dict:
        """
        Get atmospheric properties at a given altitude

        Args:
            altitude: Altitude in meters above sea level

        Returns:
            Dictionary of atmospheric properties
        """
        # Constrain altitude to reasonable values
        altitude = max(0.0, min(100000.0, altitude))

        # Calculate temperature
        if altitude <= self.h_strat:
            # Troposphere
            temperature = self.T0 - self.lapse_rate * altitude
        else:
            # Stratosphere (simplified - constant temperature)
            temperature = self.T_strat

        # Calculate pressure
        if altitude <= self.h_strat:
            # Troposphere
            exponent = 9.81 / (self.R * self.lapse_rate)
            pressure = self.p0 * (temperature / self.T0)**exponent
        else:
            # Stratosphere
            h_diff = altitude - self.h_strat
            scale_height = self.R * self.T_strat / 9.81
            p_strat = self.p0 * (self.T_strat / self.T0)**(9.81 / (self.R * self.lapse_rate))
            pressure = p_strat * np.exp(-h_diff / scale_height)

        # Calculate density
        density = pressure / (self.R * temperature)

        # Calculate speed of sound
        sound_speed = np.sqrt(self.gamma * self.R * temperature)

        return {
            'altitude': altitude,
            'temperature': temperature,
            'pressure': pressure,
            'density': density,
            'sound_speed': sound_speed
        }


class PlasmaChemistryModel:
    """Detailed plasma chemistry model for ionization calculations"""

    def __init__(self):
        """Initialize plasma chemistry model"""
        # Ionization potentials for major atmospheric constituents (in eV)
        self.ionization_potentials = {
            'N2': 15.58,
            'O2': 12.07,
            'N': 14.53,
            'O': 13.62,
            'NO': 9.26
        }

        # Mole fractions at sea level
        self.constituents = {
            'N2': 0.78084,
            'O2': 0.20946,
            'Ar': 0.00934,
            'CO2': 0.00036
        }

    def refine_electron_density(self, basic_density: float, temperature: float,
                                pressure: float, mach_number: float) -> float:
        """
        Refine electron density calculation using detailed chemistry model

        Args:
            basic_density: Basic electron density from Saha equation
            temperature: Temperature in K
            pressure: Pressure in Pa
            mach_number: Mach number

        Returns:
            Refined electron density
        """
        # This is a simplified model - a real implementation would include
        # detailed chemical kinetics calculations

        # At hypersonic speeds, dissociation of molecules occurs before ionization
        # This affects the electron density calculation
```

```
        # Correction factor based on temperature and mach number
        if temperature < 2500:
            # Not enough energy for significant ionization
            return basic_density * 0.1
        elif temperature < 6000:
            # Partial dissociation regime
            dissociation_factor = (temperature - 2500) / 3500 # 0 to 1

            # Enhanced NO production increases ionization
            no_enhancement = 1.0 + dissociation_factor * 2.0

            return basic_density * no_enhancement
        else:
            # Strong shock, with N2 and O2 dissociation and significant ionization
            # Enhanced ionization due to atomic species
            atomic_enhancement = 1.0 + (temperature - 6000) / 10000

            # Additional enhancement due to high speed
            mach_enhancement = 1.0
            if mach_number > 10:
                mach_enhancement = 1.0 + 0.1 * (mach_number - 10)

            return basic_density * atomic_enhancement * mach_enhancement


class MagneticFieldModel:
    """Model for Earth's magnetic field and its effects on plasma"""

    def __init__(self):
        """Initialize magnetic field model"""
        # Simplified dipole model parameters
        self.B0 = 3.0e-5 # Tesla, magnitude at equator
        self.earth_radius = 6371000.0 # m
        self.dipole_tilt = np.radians(11.5) # Tilt of dipole axis

    def get_magnetic_field(self, altitude: float, velocity: np.ndarray) -> np.ndarray:
        """
        Get Earth's magnetic field at specified location

        Args:
            altitude: Altitude in meters above sea level
            velocity: Vehicle velocity vector (used to estimate latitude/longitude)

        Returns:
            Magnetic field vector [Bx, By, Bz] in Tesla
        """
        # In a real implementation, this would use the vehicle's actual position
        # For this simplified model, we'll use a dipole approximation

        # Simplified position estimate (placeholder)
        # In reality, would use actual coordinates
        latitude = np.radians(30.0) # Example latitude

        # Dipole field strength scales with (R_earth / (R_earth + h))^3
        scaling = (self.earth_radius / (self.earth_radius + altitude))**3

        # Simplified dipole field calculation
        B_magnitude = self.B0 * scaling * np.sqrt(3 * np.sin(latitude)**2 + 1)

        # Direction depends on latitude
        Bx = 0.0
        By = B_magnitude * np.cos(latitude)
        Bz = 2 * B_magnitude * np.sin(latitude)

        return np.array([Bx, By, Bz])


class AblationModel:
    """Model for heat shield ablation effects on plasma sheath"""

    def __init__(self):
        """Initialize ablation model"""
        # Ablation material properties (simplified model)
        self.ablation_materials = {
            'carbon_phenolic': {
                'ablation_temp': 3000.0, # K
                'electron_factor': 2.5, # Electron density enhancement factor
                'seed_elements': ['K', 'Na']
            },
            'PICA': {
                'ablation_temp': 2500.0, # K
                'electron_factor': 1.8,
                'seed_elements': ['Si', 'C']
            }
        }

        # Default material
        self.material = 'carbon_phenolic'

    def calculate_ablation(self, velocity: float, atmos: Dict, temperature: float) -> Dict:
        """
        Calculate ablation effects on plasma sheath

        Args:
            velocity: Vehicle velocity in m/s
            atmos: Atmospheric properties
            temperature: Stagnation temperature in K

        Returns:
            Dictionary with ablation data and effects
        """
        # Check if temperature is high enough for ablation
        material_props = self.ablation_materials[self.material]

        if temperature < material_props['ablation_temp']:
            return {
                'ablating': False,
                'electron_factor': 1.0,
                'ablation_rate': 0.0
            }

        # Calculate stagnation heat flux (simplified)
        rho = atmos['density']
        heat_flux = 1.83e-8 * np.sqrt(rho / 1.0) * velocity**3 # W/m^2

        # Calculate ablation rate (simplified model)
        ablation_temp_excess = temperature - material_props['ablation_temp']
        ablation_rate = 1e-5 * ablation_temp_excess * heat_flux / 1e6 # kg/m^2/s

        # Electron density enhancement due to alkali metal impurities
        electron_factor = 1.0
        if ablation_rate > 0:
```

```
            electron_factor = material_props['
                electron_factor']

        return {
            'ablating': ablation_rate > 0,
            'electron_factor': electron_factor,
            'ablation_rate': ablation_rate,
            'heat_flux': heat_flux,
            'seed_elements': material_props['
                seed_elements']
        }


if __name__ == "__main__":
    # Example usage
    plasma_model = PlasmaSheath(
        use_detailed_model=True,
        use_magnetic_effects=True,
        consider_ablation=True
    )

    # Test different flight conditions
    test_conditions = [
        {'mach': 3.0, 'altitude': 15000, 'velocity':
            [1000.0, 0.0, 0.0]},
        {'mach': 7.0, 'altitude': 30000, 'velocity':
            [2400.0, 0.0, 0.0]},
        {'mach': 12.0, 'altitude': 50000, 'velocity':
            [4000.0, 0.0, 0.0]},
        {'mach': 20.0, 'altitude': 70000, 'velocity':
            [6000.0, 0.0, 0.0]}
    ]

    for i, condition in enumerate(test_conditions):
        print(f"\nTest condition {i+1}: Mach {
            condition['mach']} at {condition['altitude
            ']} m")

        # Calculate plasma properties
        plasma = plasma_model.
            calculate_plasma_properties(
            condition['mach'],
            condition['altitude'],
            np.array(condition['velocity'])
        )

        # Print key results
        print(f"Electron density: {plasma['
            electron_density']:.2e} m^-3")
        print(f"Plasma frequency: {plasma['
            plasma_frequency_mhz']:.2f} MHz")
        print(f"Communication blackout: {'Yes' if
            plasma['comm_blackout'] else 'No'}")

        # Print RF effects on key bands
        print("\nRF Band effects:")
        for band, effect in plasma['rf_effects'].items
            ():
            status = "BLOCKED" if effect['blocked']
                else "Usable"
            print(f" {band.upper()}: {effect['
                attenuation_db']:.2f} dB - {status}")

        # Suggest RF windows
        print("\nSuggested RF windows:")
        windows = plasma_model.suggest_rf_windows(
            plasma)
        for window in windows:
            band = window['band'].upper()
            freq_range = window['frequency_range_ghz']

            if freq_range:
                print(f" {band}: {freq_range[0]}-{
                    freq_range[1]} GHz - {window['note
                    ']}")
            else:
                print(f" {band}: {window['note']}")
```