PRD 26: KAM Theory and Planetary Stability

Pure Thought AI Challenge 26

Pure Thought AI Challenges Project

January 18, 2026

Abstract

This document presents a comprehensive Product Requirement Document (PRD) for implementing a pure-thought computational challenge. The problem can be tackled using only symbolic mathematics, exact arithmetic, and fresh code—no experimental data or materials databases required until final verification. All results must be accompanied by machine-checkable certificates.

Contents

Domain: Celestial Mechanics Dynamical Systems
Timeline: 6-9 months
Difficulty: High
Prerequisites: Hamiltonian mechanics, perturbation theory, measure theory, symplectic geometry

---

0.1 1. Problem Statement

0.1.1 Scientific Context

KAM (Kolmogorov-Arnold-Moser) theory is one of the deepest results in dynamical systems, providing a rigorous mathematical explanation for the long-term stability of planetary orbits. The classical problem dates to Newton: given N gravitating bodies with small perturbations (planet-planet interactions), do orbits remain stable forever, or do planets eventually escape or collide?

The breakthrough came in three stages:

- **Kolmogorov (1954)**: Announced that "most" invariant tori of integrable systems survive small perturbations
- **Arnold (1963)**: Proved the theorem for analytic Hamiltonians
- **Moser (1962)**: Extended to smooth (C^k)$systemswithweakerdifferentiability$

KAM Theorem (simplified): Consider a nearly-integrable Hamiltonian H = H(I) + H(I,) where H is integrable and is small. If:

- Frequencies (I) = H/I satisfy **Diophantine conditions** (non-resonant)
- H is sufficiently smooth

Then for < , there exists a **Cantor set** of invariant tori (measure → full as → 0) on which motion is **quasi-periodic** with frequencies (I).

0.1.2 Core Question

Can we numerically verify KAM conditions for realistic planetary systems and certify their long-term stability?

Key challenges:

- **Action-angle transformation**: Convert Keplerian orbits to (I,) coordinates
- **Diophantine verification**: Check |k · | /|k|^{forinfinitelymanyk}

- **KAM iteration**: Iteratively eliminate resonant terms via canonical transformations
- **Measure estimates**: Compute fraction of phase space with surviving tori
- **Solar system application**: Analyze real planetary data (Jupiter, Saturn, etc.)

0.1.3 Why This Matters

- **Planetary stability**: Rigorous proof that solar system is stable over Gyr timescales
- **Accelerator physics**: Stability of particle beams in synchrotrons
- **Plasma confinement**: Magnetic field line structure in tokamaks
- **General dynamical systems**: Paradigm for persistence of structure under perturbations
- **Chaos theory**: Boundary between regular (KAM tori) and chaotic (Arnold diffusion) motion

0.1.4 Pure Thought Advantages

KAM theory is **ideal for pure thought investigation**:

- Based on **symbolic perturbation theory** (action-angle variables)
- Diophantine conditions **verifiable algorithmically** (continued fractions)
- KAM iteration **computable via computer algebra** (Lie series)
- All results **certified via interval arithmetic** (rigorous error bounds)
- NO numerical orbit integration until verification phase
- NO empirical stability estimates

0.2 2. Mathematical Formulation

0.2.1 Integrable Systems and Invariant Tori

Integrable Hamiltonian: H(I) depends only on action variables I = (I,...,I) .
Hamilton's equations:

```
dI/dt = -   H   /      = 0    (actions constant)
d  /dt =    H   /  I   =   (I)  (angles evolve linearly)
```

Invariant tori: Phase space (I,) × foliated into n-tori I = const, each with quasi-periodic motion.

Frequencies: (I) = H/I = ((I),...,(I))

Example (Kepler problem): H = -/(2I) → = ²/I³ (single frequency, 1D torus = circle).

0.2.2 Perturbation and Resonances

Perturbed Hamiltonian: H = H(I) + H(I,) where 1.

Fourier expansion:

```
 H  (I,  ) =         H    (I) e^{ik   }
```

Resonance: Frequency vector (I) is **resonant** if k · (I) 0 for some k 0.

Small divisors problem: Perturbation series for quasi-periodic solutions involves denominators k · , which vanish at resonances → series diverges.

KAM insight: Avoid resonances by restricting to **Diophantine frequencies**.

0.2.3 Diophantine Conditions

Definition: Frequency vector is **Diophantine** with parameters (,) if:
|k · | /|k|$^{forallk \ 0}$
where |k| = |k| + ... + |k|.
Interpretation: Frequencies are "sufficiently irrational"—they avoid rational resonances by a margin that decays slower than polynomially.
Measure: Diophantine frequencies have full measure (Lebesgue) in for > n-1.
Example (golden ratio): = ((5-1)/2, 1) satisfies Diophantine conditions with = 2.

0.2.4 KAM Theorem (Precise Statement)

Theorem (Arnold 1963): Let H = H(I) + H(I,) be a real-analytic Hamiltonian on × . Assume:

- **Non-degeneracy**: det(²H/I²) 0 (frequencies change with actions)
- **Diophantine**: (I) = H/I|$_I satisfies|k\mathring{u}|/|k|^{for=n+1}$
- **Smallness**: < (depends on , , analyticity radius)

Then there exists a **Cantor set** K of actions with measure |K| → || as → 0, such that for I K:

- The invariant torus T$_I = (I,) : survivestheperturbation$
- Motion on T$_I isquasi-periodicwithfrequencies(I)$

Certificate: To certify stability, verify:

- Diophantine condition for initial frequencies
- Non-degeneracy: Hessian det 0
- Perturbation below threshold (computed via KAM estimates)

0.2.5 Certificates

All results must come with **machine-checkable certificates**:
- **Diophantine certificate**: Interval arithmetic proof that |k · | /|k|$^{for|k|K_max}$
- **Non-degeneracy certificate**: Hessian eigenvalues bounded away from zero
- **KAM convergence certificate**: Iterative scheme converges with certified error bounds
- **Measure certificate**: Lower bound on volume of surviving tori

Export format: JSON with exact algebraic numbers:

```
{
  "system": "Jupiter-Saturn",
  "frequencies": {"omega1": "2.831e-4", "omega2": "1.152e-4"},
  "diophantine_alpha": 0.001,
  "diophantine_tau": 3,
  "epsilon": 0.001,
  "kam_converged": true,
  "stable_tori_measure": 0.95,
  "certified": true
}
```

0.3 3. Implementation Approach

0.3.1 Phase 1 (Months 1-2): Action-Angle Variables

Goal: Convert Keplerian elements to action-angle coordinates.

```python
import numpy as np
import sympy as sp
from mpmath import mp
mp.dps = 100

def kepler_to_action_angle(a: float, e: float, i: float,
                           mu: float = 1.0) -> tuple:
    """
    Convert Keplerian orbital elements to Delaunay action-angle
       variables.

    Args:
        a: semi-major axis
        e: eccentricity
        i: inclination
        mu: gravitational parameter (G*M_sun)

    Returns:
        (actions, angles, frequencies)
        Actions: (L, G, H) where
          L = sqrt( a )   (mean longitude action)
          G = L*sqrt(1-e  )   (angular momentum)
          H = G*cos(i)   (vertical angular momentum)
    """
    # Delaunay actions
    L = np.sqrt(mu * a)
    G = L * np.sqrt(1 - e**2)
    H = G * np.cos(i)

    actions = np.array([L, G, H])

    # Conjugate angles: (l, g, h) where
    # l = mean anomaly
    # g = argument of perihelion
    # h = longitude of ascending node

    # Frequencies     =     H    /  I
    # For Kepler:  H    = -     /(2L  )
    omega_L = mu**2 / L**3  # Mean motion n = sqrt(   /a  )
    omega_G = 0  # Axisymmetric
    omega_H = 0  # No precession in unperturbed Kepler

    frequencies = np.array([omega_L, omega_G, omega_H])

    return actions, frequencies


def action_angle_to_cartesian(actions: np.ndarray,
                              angles: np.ndarray,
                              mu: float = 1.0) -> tuple:
    """
    Convert action-angle variables back to Cartesian positions and velocities.

    Inverse of kepler_to_action_angle.
    """
    L, G, H = actions
    l, g, h = angles

    # Reconstruct Keplerian elements
    a = L**2 / mu
    e = np.sqrt(1 - (G/L)**2)
    i = np.arccos(H/G)

    # Convert to Cartesian (standard formulas)
    # ... (omitted for brevity)

    return position, velocity


def compute_action_angle_transformation_jacobian(actions: np.ndarray) -> np.ndarray:
    """
    Compute Jacobian   (q,p)/   (  ,I) of action-angle to Cartesian transformation.

    Used for verifying symplecticity: J^T   J =   where   = [[0, I], [-I, 0]].
    """
    L, G, H = actions

    # Symbolic computation
    L_sym, G_sym, H_sym = sp.symbols('L G H', positive=True)
    l_sym, g_sym, h_sym = sp.symbols('l g h', real=True)

    # ... (compute transformation symbolically, then differentiate)

    jacobian = sp.Matrix([[...]])  # 6x6 matrix

    # Evaluate numerically
    J_numeric = np.array(jacobian.subs({L_sym: L, G_sym: G, H_sym: H}).evalf())

    return J_numeric
```

Validation: Verify transformation is canonical (symplectic) by checking J$^T J =$.

0.3.2 Phase 2 (Months 2-4): Diophantine Conditions

Goal: Verify frequency vectors satisfy Diophantine inequality.

```python
from mpmath import mp, mpf
from fractions import Fraction

def check_diophantine_condition(omega: np.ndarray,
                                alpha: float = 0.001,
                                tau: float = 3.0,
                                k_max: int = 100) -> dict:
    """
    Verify Diophantine condition |k    |        /|k|^   for all k with
       |k|     k_max.

    Returns:
        Certificate with worst-case k vector and margin.
    """
    n = len(omega)
    worst_margin = float('inf')
    worst_k = None

    for k in generate_integer_lattice(n, k_max):
        if np.all(k == 0):
            continue

        k_norm = np.sum(np.abs(k))
        k_dot_omega = abs(np.dot(k, omega))

        threshold = alpha / (k_norm ** tau)

        if k_dot_omega < threshold:
            return {
                'is_diophantine': False,
                'resonant_k': k.tolist(),
                'violation': k_dot_omega / threshold
            }

        margin = k_dot_omega / threshold
        if margin < worst_margin:
            worst_margin = margin
            worst_k = k

    return {
        'is_diophantine': True,
        'worst_k': worst_k.tolist(),
        'safety_margin': worst_margin,
        'alpha': alpha,
        'tau': tau,
        'k_max': k_max
    }


def generate_integer_lattice(n: int, k_max: int) -> list:
    """Generate all integer vectors k            with |k|     k_max."""
    from itertools import product
    vectors = []

    for k in product(range(-k_max, k_max+1), repeat=n):
        if sum(abs(ki) for ki in k) <= k_max:
            vectors.append(np.array(k))

    return vectors


def estimate_diophantine_alpha(omega: np.ndarray,
                               tau: float = 3.0,
                               k_max: int = 1000) -> float:
    """
    Estimate optimal    for given    .

    Find largest    such that Diophantine condition holds for all |k|
         k_max.
    """
    min_ratio = float('inf')

    for k in generate_integer_lattice(len(omega), k_max):
        if np.all(k == 0):
            continue

        k_norm = np.sum(np.abs(k))
        k_dot_omega = abs(np.dot(k, omega))

        ratio = k_dot_omega * (k_norm ** tau)
        if ratio < min_ratio:
            min_ratio = ratio

    alpha_optimal = min_ratio

    return alpha_optimal


def brjuno_function(omega: np.ndarray) -> float:
    """
    Compute Brjuno function B(  ) measuring how close    is to
       resonances.

    B(  ) =       log(   q      ) /  q

    where  q   are denominators in continued fraction expansion.

    KAM theorem requires B(  ) <     (weaker than Diophantine).
    """
    # Compute continued fraction for      /      (2D case)
    omega_ratio = omega[0] / omega[1]

    continued_fraction = compute_continued_fraction(omega_ratio,
       max_terms=50)

    # Compute Brjuno sum
    denominators = continued_fraction_denominators(continued_fraction)

    brjuno_sum = 0
    for n in range(len(denominators) - 1):
        q_n = denominators[n]
        q_n1 = denominators[n+1]

        brjuno_sum += np.log(q_n1) / q_n

    return brjuno_sum
```

Validation: Test on known Diophantine frequencies (golden ratio, etc.).

0.3.3 Phase 3 (Months 4-6): KAM Iteration

Goal: Implement KAM iterative scheme to construct invariant tori.

```
def kam_iteration(H0_freq: callable,
                  H1_fourier: dict,
                  epsilon: float,
                  max_iterations: int = 20,
                  tolerance: float = 1e-12) -> dict:
    """
    KAM iterative procedure to eliminate non-resonant terms.

    Algorithm (Kolmogorov):
    1. Start with H =  H   +   H
    2. Find generating function S solving homological equation {S,
      H  } =  H  ^{non-res}
    3. Apply canonical transformation via Lie series
    4. New Hamiltonian H' =  H   ' +    H    ' + O(    )
    5. Repeat until convergence

    Args:
        H0_freq: Function I        (I) giving frequencies
        H1_fourier: Dictionary {k:   H    (I)} of Fourier coefficients
        epsilon: Perturbation parameter
        max_iterations: Maximum KAM steps
        tolerance: Convergence threshold

    Returns:
        Certificate with final Hamiltonian and error estimates
    """
    # Initial data
    I0 = np.array([1.0, 0.9, 0.8])  # Reference action
    omega = H0_freq(I0)

    # Check Diophantine
    dioph_check = check_diophantine_condition(omega)
    if not dioph_check['is_diophantine']:
        return {
            'converged': False,
            'reason': 'resonance',
            'resonant_k': dioph_check['resonant_k']
        }

    # KAM iteration
    H1_current = H1_fourier.copy()
    epsilon_current = epsilon
```

```python

    for iteration in range(max_iterations):
        # Solve homological equation: i k      S    =    H
        S_fourier = {}

        for k, H1_k in H1_current.items():
            k_dot_omega = np.dot(k, omega)

            if abs(k_dot_omega) > 1e-10:  # Non-resonant
                S_fourier[k] = H1_k / (1j * k_dot_omega)

        # Compute new  H  ' via Lie series:  H  ' =  H   + {S,  H  } +
            ...
        H1_new = compute_poisson_bracket_fourier(S_fourier, H1_current,
            omega)

        # Estimate size of new perturbation
        H1_norm = sum(abs(H1_k) for H1_k in H1_new.values())

        print(f"Iteration {iteration}: || H  '|| = {H1_norm:.3e},
            = {epsilon_current**2:.3e}")

        if H1_norm < tolerance:
            return {
                'converged': True,
                'iterations': iteration,
                'final_perturbation_norm': H1_norm,
                'epsilon_effective': epsilon_current
            }

        # Update for next iteration
        H1_current = H1_new
        epsilon_current = epsilon_current ** 2  # Quadratic convergence

    return {
        'converged': False,
        'reason': 'max_iterations_reached',
        'final_perturbation_norm': H1_norm
    }


def compute_poisson_bracket_fourier(S_fourier: dict,
                                    H1_fourier: dict,
                                    omega: np.ndarray) -> dict:
    """
    Compute {S,  H  } in Fourier space.

    {S,  H  } = i          ,        (   k      )    S       H   ,
        e^{i( k   + k   )     }
    """
    result = {}

    for k1, S_k1 in S_fourier.items():
        for k2, H1_k2 in H1_fourier.items():
            k_sum = tuple(np.array(k1) + np.array(k2))

            k1_dot_omega = np.dot(k1, omega)

            term = 1j * k1_dot_omega * S_k1 * H1_k2

            if k_sum in result:
                result[k_sum] += term
            else:
                result[k_sum] = term

    return result
```

Validation: Test on pendulum (analytically solvable) and verify convergence.

0.3.4 Phase 4 (Months 6-8): Solar System Application

Goal: Apply KAM theory to analyze real planetary system stability.

```
def solar_system_kam_stability() -> dict:
    """
    Analyze KAM stability for the solar system.

    Focus on outer planets: Jupiter, Saturn, Uranus, Neptune.
    """
    # Planetary data (semi-major axis in AU, eccentricity, inclination)
    planets = {
        'Jupiter': (5.20, 0.048, 1.31),
        'Saturn': (9.54, 0.054, 2.49),
        'Uranus': (19.19, 0.047, 0.77),
        'Neptune': (30.07, 0.009, 1.77)
    }

    # Convert to action-angle
    actions = {}
    frequencies = {}

    for name, (a, e, i) in planets.items():
        I, omega = kepler_to_action_angle(a, e, np.deg2rad(i))
        actions[name] = I
        frequencies[name] = omega

    # Extract mean motions (first component of frequency vector)
    n_jupiter = frequencies['Jupiter'][0]
    n_saturn = frequencies['Saturn'][0]

    # Famous 5:2 resonance (near miss)
    resonance_ratio = n_jupiter / n_saturn
    print(f"Jupiter/Saturn frequency ratio: {resonance_ratio:.4f}
       (ideal 5:2 = {5/2})")

    # Check Diophantine for combined system
    omega_combined = np.array([frequencies[p][0] for p in
       planets.keys()])

    dioph_cert = check_diophantine_condition(omega_combined,
       alpha=1e-4, tau=4, k_max=20)

```

```python
    # Estimate perturbation strength
    epsilon = 0.001  # m_Jupiter / m_Sun ~ 10^{-3}

    # Apply KAM iteration (simplifiedwould need full perturbation Hamiltonian)
    # kam_result = kam_iteration(H0_freq, H1_fourier, epsilon)

    return {
        'planets': list(planets.keys()),
        'frequencies': {name: omega[0] for name, omega in frequencies.items()},
        'diophantine_check': dioph_cert,
        'perturbation_epsilon': epsilon,
        'conclusion': 'STABLE' if dioph_cert['is_diophantine'] else 'RESONANT'
    }


def find_resonances_in_solar_system(planets: dict,
                                    max_order: int = 10) -> list:
    """
    Find all low-order mean-motion resonances   k  n    +   k  n       0.

    Famous examples:
    - Jupiter-Saturn: 5:2 (5n_J - 2n_S      0)
    - Neptune-Pluto: 3:2
    """
    resonances = []

    planet_names = list(planets.keys())

    for i, name1 in enumerate(planet_names):
        for name2 in planet_names[i+1:]:
            n1 = planets[name1]['mean_motion']
            n2 = planets[name2]['mean_motion']

            # Search for k1, k2 such that |k1*n1 + k2*n2| < tolerance
            for k1 in range(-max_order, max_order+1):
                for k2 in range(-max_order, max_order+1):
                    if k1 == 0 and k2 == 0:
                        continue

                    resonance_value = abs(k1 * n1 + k2 * n2)

                    if resonance_value < 1e-5:  # Near resonance
                        resonances.append({
                            'planets': (name1, name2),
                            'order': (k1, k2),
                            'mismatch': resonance_value
                        })

    return resonances
```

Validation: Reproduce Laskar (1989) stability estimates for Jupiter-Saturn system.

0.3.5 Phase 5 (Months 8-9): Measure Estimates and Certificates

Goal: Compute volume of phase space occupied by KAM tori.

```python
from dataclasses import dataclass, asdict
import json

@dataclass
class KAMCertificate:
    """Complete KAM stability certificate."""

    # System identification
    system_name: str
    n_bodies: int
    perturbation_epsilon: float

    # Frequency data
    frequencies: dict
    is_diophantine: bool
    diophantine_alpha: float
    diophantine_tau: float

    # KAM iteration
    kam_converged: bool
    kam_iterations: int
    final_perturbation_norm: float

    # Measure estimates
    surviving_tori_fraction: float  # Fraction of phase space with
        stable tori

    # Stability conclusion
    is_stable: bool
    stability_timescale_years: float

    # Metadata
    computation_date: str
    precision_digits: int

    def export_json(self, filename: str):
        """Export certificate to JSON."""
        with open(filename, 'w') as f:
            json.dump(asdict(self), f, indent=2)

    def verify(self) -> bool:
        """Self-check certificate validity."""
        checks = [
            self.n_bodies > 0,
            self.perturbation_epsilon > 0,
            self.diophantine_alpha > 0,
            0 <= self.surviving_tori_fraction <= 1,
            self.stability_timescale_years > 0
        ]
        return all(checks)


def generate_kam_certificate_solar_system() -> KAMCertificate:
    """
    Generate complete KAM certificate for solar system.
    """
    stability_analysis = solar_system_kam_stability()

    cert = KAMCertificate(
        system_name='Solar System (Jupiter-Neptune)',
        n_bodies=4,
        perturbation_epsilon=0.001,
        frequencies={name: freq for name, freq in stability_analysis['frequencies'].items()},
        is_diophantine=stability_analysis['diophantine_check']['is_diophantine'],
        diophantine_alpha=stability_analysis['diophantine_check']['alpha'],
        diophantine_tau=stability_analysis['diophantine_check']['tau'],
        kam_converged=True,  # Would come from KAM iteration
        kam_iterations=15,
        final_perturbation_norm=1e-12,
        surviving_tori_fraction=0.95,  # Estimate from KAM measure theory
        is_stable=True,
        stability_timescale_years=5e9,  # Age of solar system
        computation_date='2026-01-17',
        precision_digits=100
    )

    return cert
```

Validation: Export certificates, verify all self-checks pass.

0.4 4. Example Starting Prompt

Prompt for AI System:

You are tasked with applying KAM theory to verify planetary stability. Your goals:

- **Action-Angle Transformation (Months 1-2)**:
- Convert Keplerian elements (a, e, i) to Delaunay actions (L, G, H)
- Compute frequencies = H/I
- Verify transformation is canonical (symplectic)
- **Diophantine Verification (Months 2-4)**:
- Check |k · | /|k|$^{forall|k|100}$

- Estimate optimal for Jupiter-Saturn system
- Compute Brjuno function B()
- **KAM Iteration (Months 4-6)**:
- Implement homological equation solver
- Apply Lie series canonical transformations

- Verify convergence to O(^{2}) perturbation
- **Solar System Application (Months 6-8)**:
- Analyze Jupiter, Saturn, Uranus, Neptune
- Find all resonances with order 10
- Estimate perturbation 10^{-3}
- **Certificate Generation (Months 8-9)**:
- Create KAMCertificate with all parameters
- Export to JSON with interval arithmetic bounds
- Verify stability timescale > age of solar system

Success Criteria:

- MVR (2-4 months): Action-angle for 2-body, Diophantine checks
- Strong (6-8 months): KAM iteration converges, Jupiter-Saturn analysis complete
- Publication (9 months): Full solar system certificate, measure estimates

References:

- Arnold (1963): Proof of KAM theorem
- Laskar (1989): Numerical chaos in solar system
- Celletti Chierchia (2007): KAM stability for realistic models

Begin by implementing action-angle transformation for Jupiter orbit.

0.5 5. Success Criteria

0.5.1 Minimum Viable Result (Months 1-4)

Core Achievements:

- Action-angle transformation for Kepler problem
- Diophantine verification for 2D frequency vectors
- Basic KAM iteration (3-5 steps) for toy Hamiltonian
- Certificate generation framework

Validation:

- Canonical transformation verified (Jacobian check)
- Diophantine condition tested on golden ratio

- KAM iteration reduces perturbation by factor 100

Deliverables:

- Python module `kam`$_theory.py$
- Jupyter notebook: Jupiter-Saturn resonance analysis
- JSON certificate for simple 2-body system

0.5.2 Strong Result (Months 4-8)

Extended Capabilities:

- Full KAM iteration with 10+ steps
- Solar system stability analysis (Jupiter-Neptune)
- Resonance finding algorithm
- Measure estimates: fraction of surviving tori
- Comparison to Laskar (1989) results

Publications Benchmark:

- Reproduce Laskar stability timescales
- Match Diophantine parameters to within 10

Deliverables:

- Database of certificates for 10+ planetary configurations
- Resonance map (frequency space plot)
- Stability report: timescales vs perturbation strength

0.5.3 Publication-Quality Result (Months 8-9)

Novel Contributions:

- Rigorous error bounds on KAM iteration
- Optimal Diophantine parameters for solar system
- Extension to 3-body resonances (secular dynamics)
- Formal verification: Coq/Lean proofs of key lemmas
- Interactive visualization: invariant tori in phase space

Beyond Literature:

- Improve KAM convergence rates
- Discover new stability islands in phase space

- Apply to exoplanetary systems

Deliverables:

- Arxiv preprint: "Certified KAM Stability for the Solar System"
- GitHub repository with all code and certificates
- Web tool: check KAM stability for arbitrary planetary systems

---

0.6 6. Verification Protocol

```python
def verify_kam_certificate(cert: KAMCertificate) -> dict:
    """
    Automated verification of KAM certificate.
    """
    results = {}

    # Check 1: Diophantine condition
    omega_array = np.array(list(cert.frequencies.values()))
    dioph_recheck = check_diophantine_condition(omega_array,
        cert.diophantine_alpha, cert.diophantine_tau)
    results['diophantine_verified'] =
        dioph_recheck['is_diophantine']

    # Check 2: KAM convergence
    results['kam_converged'] = cert.kam_converged

    # Check 3: Measure estimate
    results['measure_reasonable'] = (0.5 <
        cert.surviving_tori_fraction <= 1.0)

    # Check 4: Stability conclusion
    results['stability_consistent'] = (
        cert.is_stable == (cert.is_diophantine and
            cert.kam_converged)
    )

    # Overall verdict
    results['all_checks_passed'] = all(
        v for v in results.values() if isinstance(v, bool)
    )

    return results
```

0.7 7. Resources and Milestones

0.7.1 Essential References

- **Foundational Papers**:

- Kolmogorov (1954): "On conservation of conditionally periodic motions"
- Arnold (1963): "Proof of A.N. Kolmogorov's theorem"
- Moser (1962): "On invariant curves of area-preserving mappings"
- **Modern Developments**:
- Celletti Chierchia (2007): "KAM stability and celestial mechanics"
- Laskar (1989): "A numerical experiment on the chaotic behaviour of the Solar System"
- Féjoz (2004): "Démonstration du 'théorème d'Arnold' sur la stabilité du système planétaire"
- **Textbooks**:
- Arnold (1989): *Mathematical Methods of Classical Mechanics*
- Broer Sevryuk (2007): "KAM theory: quasi-periodicity in dynamical systems"

0.7.2 Milestone Checklist

Month 1: Action-angle transformation implemented

Month 2: Diophantine verifier working for n 4

Month 3: KAM iteration converges for pendulum

Month 4: Jupiter-Saturn frequencies computed

Month 5: Diophantine verified for solar system

Month 6: KAM iteration for planetary Hamiltonian

Month 7: Resonance map generated

Month 8: Measure estimates computed

Month 9: Full certificate database exported

End of PRD 26