

Nitrogen Balance Model - Crop Physiologist Expert Guide

Physiological Foundation of Plant Nitrogen Balance

Nitrogen: The Master Nutrient

As a crop physiologist, nitrogen is the most critical nutrient for plant growth and productivity. It's the cornerstone of proteins, nucleic acids, chlorophyll, and countless metabolic compounds.

Understanding nitrogen balance requires appreciating the complex interplay between uptake, assimilation, transport, and remobilization.

The Nitrogen Cycle in Plants

1. Uptake Phase

- **Primary forms:** NO_3^- , NH_4^+ , amino acids, urea
- **Location:** Root epidermis and cortex
- **Mechanism:** Active transport (H^+ -coupled symporters)
- **Energy cost:** 2-4 ATP per ion

2. Assimilation Phase

- **NO_3^- reduction:** $\text{NO}_3^- \rightarrow \text{NO}_2^- \rightarrow \text{NH}_4^+$ (via nitrate reductase, nitrite reductase)
- **NH_4^+ assimilation:** $\text{NH}_4^+ + \text{glutamate} \rightarrow \text{glutamine}$ (via glutamine synthetase)
- **Location:** Roots and shoots (species-dependent)
- **Energy cost:** 12-16 ATP per NH_4^+ assimilated

3. Transport Phase

- **Xylem:** NO_3^- , amino acids (asparagine, glutamine)
- **Phloem:** Amino acids, proteins
- **Direction:** Bidirectional (source-sink relationships)

4. Remobilization Phase

- **Senescence-induced:** Old tissues \rightarrow young tissues
- **Stress-induced:** Any tissue \rightarrow priority sinks
- **Efficiency:** 60-80% of nitrogen recoverable

Mathematical Framework

Nitrogen Balance Equation

python

```
def calculate_nitrogen_balance(uptake_rate, assimilation_rate, growth_demand,
                               remobilization_rate, losses):
    """
    Fundamental nitrogen balance equation

    
$$dN/dt = \text{Uptake} + \text{Remobilization} - \text{Growth\_demand} - \text{Losses}$$


    Where:
    - Uptake: Fresh nitrogen from solution
    - Remobilization: Recycled nitrogen from senescence
    - Growth_demand: Nitrogen needed for new growth
    - Losses: Exudation, volatilization
    """

    nitrogen_gain = uptake_rate + remobilization_rate
    nitrogen_demand = growth_demand + losses

    net_balance = nitrogen_gain - nitrogen_demand

    return {
        'net_balance': net_balance,
        'nitrogen_status': classify_nitrogen_status(net_balance, growth_demand),
        'supply_sufficiency': nitrogen_gain / nitrogen_demand if nitrogen_demand > 0 else 1.0
    }

def classify_nitrogen_status(net_balance, growth_demand):
    """Classify plant nitrogen status"""
    if net_balance >= 0.1 * growth_demand:
        return 'luxury_consumption'
    elif net_balance >= 0:
        return 'sufficient'
    elif net_balance >= -0.2 * growth_demand:
        return 'marginal'
    else:
        return 'deficient'
```

Multi-Form Nitrogen Uptake

python

```
@dataclass
```

```
class NitrogenUptakeKinetics:
```

```
    """Kinetic parameters for different nitrogen forms"""
```

```
    # Uptake parameters ( $\mu\text{mol g}^{-1} \text{root h}^{-1}$ )
```

```
    uptake_kinetics: Dict[str, Dict[str, float]] = field(default_factory=lambda: {
```

```
        'NO3': {
```

```
            'vmax': 15.0,      # High capacity system
```

```
            'km': 50.0,       #  $\mu\text{M}$  half-saturation
```

```
            'energy_cost': 2.0 # ATP per ion
```

```
        },
```

```
        'NH4': {
```

```
            'vmax': 8.0,      # Lower capacity (toxicity)
```

```
            'km': 20.0,       # Higher affinity
```

```
            'energy_cost': 1.0 # Lower immediate cost
```

```
        },
```

```
        'AA': {
```

```
            'vmax': 5.0,      # Specialized transporters
```

```
            'km': 10.0,       # Very high affinity
```

```
            'energy_cost': 1.5 # Moderate cost
```

```
        },
```

```
        'UREA': {
```

```
            'vmax': 3.0,      # Limited capacity
```

```
            'km': 15.0,       # Moderate affinity
```

```
            'energy_cost': 1.0 # Low initial cost
```

```
        }
```

```
    })
```

```
def calculate_multi_form_uptake(solution_concentrations, root_mass, kinetics,
                                environmental_factors):
```

```
    """
```

```
    Calculate uptake of multiple nitrogen forms with competition
```

```
    Physiological considerations:
```

- Transporter competition (NO_3^- vs Cl^-)
- Metabolic regulation (feedback inhibition)
- Environmental modulation (temperature, pH)

```
    """
```

```
    total_uptake = 0.0
```

```
    uptake_breakdown = {}
```

```
    for n_form, concentration in solution_concentrations.items():
```

```

if n_form in kinetics.uptake_kinetics:
    params = kinetics.uptake_kinetics[n_form]

    # Basic Michaelis-Menten uptake
    base_uptake = (
        params['vmax'] * concentration /
        (params['km'] + concentration)
    )

    # Environmental modifications
    temp_factor = environmental_factors.get('temperature_factor', 1.0)
    ph_factor = environmental_factors.get('ph_factor', 1.0)
    water_factor = environmental_factors.get('water_status', 1.0)

    # Competitive inhibition (simplified)
    competition_factor = calculate_competition_effect(n_form, solution_concentrations)

    # Final uptake rate
    actual_uptake = (
        base_uptake *
        root_mass *
        temp_factor *
        ph_factor *
        water_factor *
        competition_factor
    )

    total_uptake += actual_uptake
    uptake_breakdown[n_form] = {
        'uptake_rate': actual_uptake,
        'concentration': concentration,
        'efficiency': actual_uptake / (base_uptake * root_mass) if base_uptake > 0 else 0
    }

return total_uptake, uptake_breakdown

```

```

def calculate_competition_effect(target_form, all_concentrations):
    """Calculate competitive inhibition between nitrogen forms"""

```

```

# Known competitive interactions
competition_matrix = {
    'NO3': {'Cl': 0.3, 'SO4': 0.1}, # NO3- competes with Cl-, SO42-
    'NH4': {'K': 0.2, 'Ca': 0.1}, # NH4+ competes with K+, Ca2+
    'AA': {}, # Amino acids have specific transporters

```

```

    'UREA': {}                # Urea has specific transporters
}

if target_form not in competition_matrix:
    return 1.0

competition_effect = 1.0
competitors = competition_matrix[target_form]

for competitor, inhibition_strength in competitors.items():
    competitor_conc = all_concentrations.get(competitor, 0)
    if competitor_conc > 0:
        # Simple competitive inhibition model
        competition_effect *= (1.0 - inhibition_strength * min(1.0, competitor_conc / 100))

return max(0.1, competition_effect) # Minimum 10% activity

```

Nitrogen Assimilation Pathways

Nitrate Reduction Pathway

python

```
def nitrate_assimilation_pathway(no3_uptake, tissue_type, light_status, temperature):
```

```
    """
```

```
    Model nitrate reduction and assimilation
```

```
    Pathway:  $\text{NO}_3^- \rightarrow \text{NO}_2^- \rightarrow \text{NH}_4^+ \rightarrow \text{Glutamine} \rightarrow \text{other amino acids}$ 
```

```
    Enzyme regulation:
```

- Nitrate reductase: Light-activated, substrate-induced
- Nitrite reductase: Constitutive, ferredoxin-dependent
- Glutamine synthetase: NH_4^+ -induced, feedback regulated

```
    """
```

```
    # Nitrate reductase activity (tissue and light dependent)
```

```
    if tissue_type == 'leaves':
```

```
        if light_status == 'light':
```

```
            nr_activity = 1.0 # Full activity in light
```

```
        else:
```

```
            nr_activity = 0.3 # Reduced activity in dark
```

```
    else: # roots
```

```
        nr_activity = 0.6 # Constitutive root activity
```

```
    # Temperature effect on enzyme activity
```

```
    temp_factor = 2.0 ** ((temperature - 25) / 10)
```

```
    temp_factor = min(temp_factor, 3.0) # Cap at 3x
```

```
    #  $\text{NO}_3^-$  reduction rate
```

```
    no3_reduction_rate = no3_uptake * nr_activity * temp_factor
```

```
    #  $\text{NO}_2^-$  reduction (faster than  $\text{NO}_3^-$  reduction, not limiting)
```

```
    no2_reduction_rate = no3_reduction_rate * 1.2
```

```
    #  $\text{NH}_4^+$  assimilation (glutamine synthetase)
```

```
    nh4_available = min(no2_reduction_rate, no3_reduction_rate) # Stoichiometric
```

```
    # Glutamine synthesis capacity
```

```
    gs_activity = calculate_gs_activity(tissue_type, nh4_available)
```

```
    glutamine_synthesis = min(nh4_available, gs_activity)
```

```
    # Energy costs
```

```
    energy_cost = {
```

```
        'no3_reduction': no3_reduction_rate * 8, # 8 ATP per  $\text{NO}_3^-$ 
```

```
        'nh4_assimilation': glutamine_synthesis * 1 # 1 ATP per glutamine
```

```
    }
```

```

return {
    'glutamine_production': glutamine_synthesis,
    'nh4_excess': nh4_available - glutamine_synthesis,
    'energy_cost': sum(energy_cost.values()),
    'nr_activity': nr_activity,
    'pathway_efficiency': glutamine_synthesis / no3_uptake if no3_uptake > 0 else 0
}

def calculate_gs_activity(tissue_type, nh4_concentration):
    """
    Calculate glutamine synthetase activity

    Regulation:
    - Substrate induction by NH4+
    - Product inhibition by glutamine
    - Tissue-specific expression levels
    """

    # Base activity by tissue type
    base_activity = {
        'roots': 1.0,    # High root GS activity
        'leaves': 0.8,   # Moderate leaf activity
        'stems': 0.3     # Low stem activity
    }.get(tissue_type, 0.5)

    # Substrate induction (Hill equation)
    km_nh4 = 0.1 # mM
    hill_coefficient = 2.0
    substrate_factor = (
        nh4_concentration ** hill_coefficient /
        (km_nh4 ** hill_coefficient + nh4_concentration ** hill_coefficient)
    )

    return base_activity * substrate_factor

```

Amino Acid Biosynthesis Network

python

```
def amino_acid_biosynthesis(glutamine_pool, carbon_skeletons, energy_status):
```

```
    """
```

```
    Model amino acid biosynthesis from glutamine and carbon skeletons
```

```
    Key pathways:
```

1. Glutamine → Glutamate (glutamate synthase)
2. Glutamate → other amino acids (transaminases)
3. Branched-chain amino acids (pyruvate family)
4. Aromatic amino acids (shikimate pathway)

```
    """
```

```
    # Available carbon skeletons for amino acid synthesis
```

```
    carbon_availability = {
```

```
        'pyruvate_family': carbon_skeletons.get('pyruvate', 0),    # Ala, Val, Leu, Ile
```

```
        'oxaloacetate_family': carbon_skeletons.get('oxaloacetate', 0), # Asp, Asn, Thr, Met, Lys
```

```
        'alpha_ketoglutarate_family': carbon_skeletons.get('alpha_kg', 0), # Glu, Gln, Pro, Arg
```

```
        'aromatic_family': carbon_skeletons.get('pep', 0),        # Phe, Tyr, Trp
```

```
        'serine_family': carbon_skeletons.get('3pg', 0)           # Ser, Gly, Cys
```

```
    }
```

```
    # Amino acid synthesis rates (limited by carbon or nitrogen)
```

```
    amino_acid_production = {}
```

```
    total_nitrogen_used = 0
```

```
    for family, carbon_supply in carbon_availability.items():
```

```
        # Nitrogen demand for this family
```

```
        family_n_demand = calculate_family_n_demand(family)
```

```
        # Available nitrogen (from glutamine pool)
```

```
        available_n = min(glutamine_pool - total_nitrogen_used, family_n_demand)
```

```
        available_n = max(0, available_n)
```

```
        # Synthesis rate (limited by carbon or nitrogen)
```

```
        if carbon_supply > 0 and available_n > 0:
```

```
            synthesis_rate = min(carbon_supply, available_n) * energy_status
```

```
            amino_acid_production[family] = synthesis_rate
```

```
            total_nitrogen_used += synthesis_rate
```

```
        else:
```

```
            amino_acid_production[family] = 0
```

```
    # Calculate protein synthesis potential
```

```
    protein_synthesis_potential = min(amino_acid_production.values()) if amino_acid_production else 0
```



```

return {
    'amino_acid_production': amino_acid_production,
    'total_nitrogen_used': total_nitrogen_used,
    'nitrogen_remaining': max(0, glutamine_pool - total_nitrogen_used),
    'protein_synthesis_potential': protein_synthesis_potential,
    'carbon_limitation': sum(1 for c in carbon_availability.values() if c == 0),
    'nitrogen_limitation': 1 if total_nitrogen_used >= glutamine_pool else 0
}

```

```

def calculate_family_n_demand(amino_acid_family):
    """Calculate nitrogen demand for each amino acid family"""

    # Nitrogen atoms per amino acid in each family (weighted average)
    family_n_content = {
        'pyruvate_family': 1.0,      # Mostly 1 N per amino acid
        'oxaloacetate_family': 1.2,  # Some have 2 N (Asn, Lys)
        'alpha_ketoglutarate_family': 1.8, # Pro, Arg have more N
        'aromatic_family': 1.0,      # 1 N per amino acid
        'serine_family': 1.0         # 1 N per amino acid
    }

    return family_n_content.get(amino_acid_family, 1.0)

```

Nitrogen Transport and Allocation

Source-Sink Nitrogen Transport

python

```
def nitrogen_transport_allocation(organ_n_status, growth_rates, transport_capacity):
```

```
    """
```

```
    Model nitrogen transport between plant organs
```

```
    Transport forms:
```

- Xylem: NO_3^- , amino acids (source: roots)
- Phloem: amino acids, proteins (bidirectional)

```
    Allocation priority:
```

1. Active meristems (high growth rate)
2. Young expanding leaves
3. Reproductive organs
4. Storage organs

```
    """
```

```
    # Calculate nitrogen demands by organ
```

```
    organ_demands = {}
```

```
    for organ, growth_rate in growth_rates.items():
```

```
        # N demand based on growth rate and tissue N concentration
```

```
        target_n_concentration = get_target_n_concentration(organ)
```

```
        n_demand = growth_rate * target_n_concentration
```

```
        organ_demands[organ] = n_demand
```

```
    # Calculate available nitrogen for transport
```

```
    available_sources = {}
```

```
    for organ, n_status in organ_n_status.items():
```

```
        if n_status['current_n'] > n_status['minimal_n']:
```

```
            # Excess nitrogen available for export
```

```
            exportable_n = n_status['current_n'] - n_status['minimal_n']
```

```
            # Transport capacity limitation
```

```
            actual_export = min(exportable_n, transport_capacity.get(organ, 0))
```

```
            available_sources[organ] = actual_export
```

```
    # Allocation algorithm (priority-based)
```

```
    organ_priorities = {
```

```
        'shoot_meristem': 10,
```

```
        'young_leaves': 8,
```

```
        'expanding_leaves': 6,
```

```
        'mature_leaves': 3,
```

```
        'roots': 5,
```

```
        'reproductive_organs': 9,
```

```
        'storage_organs': 2
```

```
    }
```

```

# Sort organs by priority
sorted_demands = sorted(
    organ_demands.items(),
    key=lambda x: organ_priorities.get(x[0], 5),
    reverse=True
)

total_available = sum(available_sources.values())
allocations = {}
remaining_supply = total_available

for organ, demand in sorted_demands:
    if remaining_supply > 0:
        allocated = min(demand, remaining_supply)
        allocations[organ] = allocated
        remaining_supply -= allocated
    else:
        allocations[organ] = 0

return {
    'allocations': allocations,
    'total_demand': sum(organ_demands.values()),
    'total_supply': total_available,
    'supply_demand_ratio': total_available / sum(organ_demands.values()) if organ_demands else 1.0,
    'unmet_demand': max(0, sum(organ_demands.values()) - total_available)
}

def get_target_n_concentration(organ_type):
    """Get target nitrogen concentration for different organs"""

    # Target N concentrations (% dry weight)
    targets = {
        'young_leaves': 0.055,    # High N for photosynthesis
        'mature_leaves': 0.045,   # Standard N concentration
        'old_leaves': 0.035,      # Lower N (remobilization)
        'stems': 0.020,           # Structural tissue
        'roots': 0.028,           # Moderate N for uptake
        'reproductive_organs': 0.060, # High N demand
        'storage_organs': 0.015    # Low N in storage
    }

    return targets.get(organ_type, 0.030) # Default 3%

```

Critical Nitrogen Concentrations

python

```

def calculate_critical_n_curve(plant_biomass, growth_stage):
    """
    Calculate critical nitrogen concentration using allometric relationships

    Based on:  $N_c = a * W^{-b}$ 
    Where:  $N_c$  = critical N concentration,  $W$  = plant dry weight

    Research basis: Lemaire & Gastal (1997), Greenwood et al. (1990)
    """

    # Species and stage-specific parameters
    stage_parameters = {
        'vegetative': {'a': 5.7, 'b': 0.442},
        'reproductive': {'a': 5.2, 'b': 0.380},
        'senescence': {'a': 4.8, 'b': 0.350}
    }

    params = stage_parameters.get(growth_stage, stage_parameters['vegetative'])

    # Critical N concentration (% dry weight)
    critical_n = params['a'] * (plant_biomass ** -params['b'])

    # Minimum N concentration (below which death occurs)
    minimum_n = critical_n * 0.6

    # Luxury N concentration (above which no growth benefit)
    luxury_n = critical_n * 1.4

    return {
        'critical_n': critical_n,
        'minimum_n': minimum_n,
        'luxury_n': luxury_n,
        'deficiency_threshold': critical_n * 0.8,
        'sufficiency_threshold': critical_n * 1.1
    }

def nitrogen_stress_factor(current_n, critical_n_curve):
    """
    Calculate nitrogen stress factor based on current vs critical N

    Stress factor: 1.0 = no stress, 0.0 = severe stress
    """

```

```
current_conc = current_n
critical_conc = critical_n_curve['critical_n']
minimum_conc = critical_n_curve['minimum_n']

if current_conc >= critical_conc:
    # Sufficient nitrogen
    stress_factor = 1.0
elif current_conc >= minimum_conc:
    # Deficient but not lethal
    stress_factor = (current_conc - minimum_conc) / (critical_conc - minimum_conc)
else:
    # Severe deficiency
    stress_factor = 0.1 * (current_conc / minimum_conc) if minimum_conc > 0 else 0

return max(0.0, min(1.0, stress_factor))
```

Nitrogen Remobilization

Senescence-Induced Remobilization

python

```
def senescence_nitrogen_remobilization(senescing_biomass, tissue_n_content,
                                       remobilization_efficiency):
```

```
    """
```

Calculate nitrogen remobilization during leaf senescence

Process:

1. Protein degradation (proteases activated)
2. Amino acid export via phloem
3. Structural N remains in senescing tissue

Efficiency varies by:

- Nutrient status (higher under deficiency)
- Environmental stress
- Genetic factors

```
    """
```

Nitrogen pools in senescing tissue

total_n = tissue_n_content

Categorize nitrogen by mobility

```
nitrogen_pools = {
    'protein_n': total_n * 0.70,    # Highly mobile (enzymes, RuBisCO)
    'nucleic_acid_n': total_n * 0.15, # Moderately mobile
    'chlorophyll_n': total_n * 0.10, # Mobile (Mg2+ reused too)
    'structural_n': total_n * 0.05   # Immobile (cell walls)
}
```

Remobilization efficiency by pool

```
pool_efficiencies = {
    'protein_n': remobilization_efficiency * 0.9,    # 90% of base efficiency
    'nucleic_acid_n': remobilization_efficiency * 0.7, # 70% of base efficiency
    'chlorophyll_n': remobilization_efficiency * 0.8, # 80% of base efficiency
    'structural_n': 0.1                               # Only 10% mobile
}
```

Calculate remobilized nitrogen by pool

```
remobilized_by_pool = {}
total_remobilized = 0
```

```
for pool, n_amount in nitrogen_pools.items():
    pool_efficiency = pool_efficiencies[pool]
    pool_remobilized = n_amount * pool_efficiency
    remobilized_by_pool[pool] = pool_remobilized
```

```
total_remobilized += pool_remobilized
```

```
# Remaining nitrogen in senescing tissue
```

```
remaining_n = total_n - total_remobilized
```

```
return {
```

```
    'total_remobilized': total_remobilized,
```

```
    'remobilization_efficiency': total_remobilized / total_n if total_n > 0 else 0,
```

```
    'remobilized_by_pool': remobilized_by_pool,
```

```
    'remaining_n': remaining_n,
```

```
    'n_recovery_percentage': (total_remobilized / total_n) * 100 if total_n > 0 else 0
```

```
}
```

```
def stress_induced_remobilization(stress_factors, organ_n_pools, sink_demands):
```

```
    """
```

```
    Model nitrogen remobilization under stress conditions
```

```
    Stress triggers:
```

```
    - Water stress (drought)
```

```
    - Temperature stress (heat/cold)
```

```
    - Light stress (shading)
```

```
    - Root damage
```

```
    """
```

```
# Calculate overall stress level
```

```
stress_weights = {
```

```
    'water_stress': 0.4,
```

```
    'temperature_stress': 0.3,
```

```
    'light_stress': 0.2,
```

```
    'root_stress': 0.1
```

```
}
```

```
overall_stress = sum(
```

```
    stress_factors.get(stress_type, 0) * weight
```

```
    for stress_type, weight in stress_weights.items()
```

```
)
```

```
# Stress-induced remobilization only occurs above threshold
```

```
stress_threshold = 0.3
```

```
if overall_stress < stress_threshold:
```

```
    return {'remobilized_n': 0, 'stress_level': overall_stress}
```

```
# Enhanced remobilization under stress
```

```
stress_factor = (overall_stress - stress_threshold) / (1.0 - stress_threshold)
```



```

# Priority organs for nitrogen allocation under stress
stress_priorities = {
    'shoot_meristem': 1.0,    # Highest priority
    'young_leaves': 0.8,     # High priority
    'roots': 0.6,            # Moderate priority (survive stress)
    'old_leaves': 0.1,       # Lowest priority (sacrifice first)
    'stems': 0.3             # Low priority
}

# Calculate nitrogen mobilization from low-priority organs
total_mobilized = 0
mobilization_details = {}

for organ, n_pool in organ_n_pools.items():
    organ_priority = stress_priorities.get(organ, 0.5)

    # Low priority organs donate nitrogen
    if organ_priority < 0.5:
        mobilizable_fraction = stress_factor * (0.5 - organ_priority)
        mobilized_n = n_pool * mobilizable_fraction
        total_mobilized += mobilized_n
        mobilization_details[organ] = {
            'mobilized': mobilized_n,
            'remaining': n_pool - mobilized_n,
            'priority': organ_priority
        }

return {
    'total_mobilized': total_mobilized,
    'stress_level': overall_stress,
    'mobilization_details': mobilization_details,
    'emergency_response': overall_stress > 0.7
}

```

Practical Applications

Nitrogen Management Strategies

python

```
def optimize_nitrogen_supply(growth_stage, environmental_conditions, target_yield):
```

```
    """
```

```
    Optimize nitrogen supply strategy for hydroponic systems
```

```
    Considerations:
```

1. Growth stage requirements
2. Environmental N use efficiency
3. Economic optimization
4. Environmental impact

```
    """
```

```
    # Base nitrogen requirements by growth stage (ppm in solution)
```

```
    base_requirements = {
```

```
        'seedling': {'NO3': 80, 'NH4': 15},
```

```
        'vegetative': {'NO3': 150, 'NH4': 20},
```

```
        'reproductive': {'NO3': 120, 'NH4': 15},
```

```
        'maturation': {'NO3': 100, 'NH4': 10}
```

```
    }
```

```
    base_n = base_requirements.get(growth_stage, base_requirements['vegetative'])
```

```
    # Environmental adjustments
```

```
    adjustments = {}
```

```
    # Temperature effects
```

```
    if environmental_conditions['temperature'] > 25:
```

```
        adjustments['temp_factor'] = 1.1 # Higher metabolism
```

```
    elif environmental_conditions['temperature'] < 18:
```

```
        adjustments['temp_factor'] = 0.9 # Slower metabolism
```

```
    else:
```

```
        adjustments['temp_factor'] = 1.0
```

```
    # Light effects
```

```
    light_level = environmental_conditions.get('daily_light_integral', 15)
```

```
    if light_level > 20:
```

```
        adjustments['light_factor'] = 1.15 # Higher photosynthesis
```

```
    elif light_level < 12:
```

```
        adjustments['light_factor'] = 0.85 # Lower photosynthesis
```

```
    else:
```

```
        adjustments['light_factor'] = 1.0
```

```
    # pH effects on availability
```

```
    ph = environmental_conditions.get('ph', 6.0)
```

```

if 5.8 <= ph <= 6.2:
    adjustments['ph_factor'] = 1.0
else:
    adjustments['ph_factor'] = 0.9

# Calculate optimized concentrations
combined_factor = (
    adjustments['temp_factor'] *
    adjustments['light_factor'] *
    adjustments['ph_factor']
)

optimized_n = {
    'NO3': base_n['NO3'] * combined_factor,
    'NH4': base_n['NH4'] * combined_factor
}

# Safety bounds
optimized_n['NO3'] = max(50, min(200, optimized_n['NO3']))
optimized_n['NH4'] = max(5, min(30, optimized_n['NH4']))

return {
    'optimized_concentrations': optimized_n,
    'adjustment_factors': adjustments,
    'total_n': optimized_n['NO3'] + optimized_n['NH4'],
    'no3_nh4_ratio': optimized_n['NO3'] / optimized_n['NH4']
}

def nitrogen_use_efficiency_analysis(n_supplied, n_uptake, biomass_production, n_export):
    """
    Analyze nitrogen use efficiency metrics

    Key metrics:
    - NUE (Nitrogen Use Efficiency): biomass/N_uptake
    - NUtE (N Utilization Efficiency): biomass/plant_N
    - NUpE (N Uptake Efficiency): N_uptake/N_supplied
    """

    # Basic efficiency calculations
    nupe = n_uptake / n_supplied if n_supplied > 0 else 0 # Uptake efficiency
    nute = biomass_production / n_uptake if n_uptake > 0 else 0 # Utilization efficiency
    nue = nupe * nute # Overall use efficiency

    # N recovery efficiency (including remobilization)

```

```

n_recovered = n_export # N in harvested biomass
recovery_efficiency = n_recovered / n_supplied if n_supplied > 0 else 0

# Classification of efficiency
efficiency_class = classify_nue_performance(nue)

return {
    'nitrogen_uptake_efficiency': nupe,
    'nitrogen_utilization_efficiency': nute,
    'nitrogen_use_efficiency': nue,
    'nitrogen_recovery_efficiency': recovery_efficiency,
    'efficiency_class': efficiency_class,
    'biomass_per_n': biomass_production / n_uptake if n_uptake > 0 else 0
}

def classify_nue_performance(nue_value):
    """Classify nitrogen use efficiency performance"""
    if nue_value > 50:
        return 'excellent'
    elif nue_value > 40:
        return 'good'
    elif nue_value > 30:
        return 'moderate'
    elif nue_value > 20:
        return 'poor'
    else:
        return 'very_poor'

```

This nitrogen balance model provides the foundation for optimizing nitrogen nutrition in hydroponic systems, enabling precise control of plant nutrition for maximum productivity and resource efficiency.