energetium

Calculate chemistry energetics and kinetics equations with WebAssembly acceleration.

v0.0.1 November 25, 2025

https://github.com/Typsium/Energetium

Typsium Community
https://github.com/Typsium

β-吲哚基丙氨酸
https://github.com/tryptophawa

Contents

Energetium

analyze-kinetics

Analyze reaction kinetics with multiple parameters

Arguments:

- a: Pre-exponential factor
- ea: Activation energy (kJ/mol)
- temp: Temperature (K)
- order: Reaction order
- initial-conc: Initial concentration (optional)
- precision: Number of decimal places
- scientific: Scientific notation mode

Returns: Dictionary with all kinetic parameters

Parameters

```
analyze-kinetics(
  a,
  ea,
  temp,
  order,
  initial-conc,
  precision,
  scientific
)
```

analyze-reaction

Complete reaction analysis

Calculates ΔH, ΔS, and ΔG for a reaction and formats the results

Arguments:

- reactants: Array of tuples (formula, coefficient)
- products: Array of tuples (formula, coefficient)
- temp: Temperature in Kelvin (default: 298.15 K)
- data: Optional custom thermodynamic data dictionary
- precision: Number of decimal places (default: 2)
- scientific: Format mode (default: auto)

Returns: Dictionary with all calculated values

Parameters

```
analyze-reaction(
  reactants,
  products,
  temp,
  data,
  precision,
  scientific
) -> dict
```

calc-activation-energy

Calculate activation energy from rate constants at two temperatures Ea = R·ln(k2/k1) / (1/T1 - 1/T2)

Arguments:

- k1: Rate constant at temperature T1
- t1: Temperature 1 (K)
- k2: Rate constant at temperature T2
- t2: Temperature 2 (K)

Returns: Dictionary with activation energy in kJ/mol

Example:

```typst
1 #let ea = calc-activation-energy(0.001, 300, 0.01, 350)
```

Parameters

```
calc-activation-energy(
  k1,
  t1,
  k2,
  t2
)
```

calc-equilibrium-constant

Calculate equilibrium constant from Gibbs free energy K = exp(-ΔG / RT)

Arguments:

- gibbs-energy: Gibbs free energy change in kJ/mol
- temp: Temperature in Kelvin (default: 298.15 K)

Returns: Dictionary with keys `value` (number) and `unit` (string)

Parameters

```
calc-equilibrium-constant(
  gibbs-energy,
  temp
) -> dict
```

calc-gibbs-energy

Calculate Gibbs free energy change ΔG = ΔH - T·ΔS

Arguments:

- enthalpy: Enthalpy change in kJ/mol
- entropy: Entropy change in J/(mol·K)
- temp: Temperature in Kelvin (default: 298.15 K)

Returns: Dictionary with keys `value` (number) and `unit` (string)

Parameters

```
calc-gibbs-energy(
  enthalpy,
  entropy,
  temp
) -> dict
```

calc-half-life

Calculate half-life for a reaction

- Zero order: t_1/2 = [A]0 / (2k)
- First order: t_1/2 = ln(2) / k
- Second order: t_1/2 = 1 / (k[A]0)

Arguments:

- k: Rate constant
- order: Reaction order (0, 1, or 2, default: 1)
- initial-conc: Initial concentration (default: 1.0, required for 0th and 2nd order)

Returns: Dictionary with half-life in seconds

Example:

```
1 #let t-half = calc-half-life(0.693, order: 1)                    typst
```

Parameters

```
calc-half-life(
  k,
  order,
  initial-conc
)
```

calc-rate-constant-arrhenius

Calculate rate constant using Arrhenius equation k = A·exp(-Ea/(R·T))

Arguments:

- a: Pre-exponential factor (frequency factor)
- ea: Activation energy (kJ/mol)

- temp: Temperature (K, default: 298.15)

Returns: Dictionary with rate constant value and unit

Example:

```typst
#let k = calc-rate-constant-arrhenius(1e13, 50, temp: 298.15)
```

Parameters

```
calc-rate-constant-arrhenius(
  a,
  ea,
  temp
)
```

calc-rate-constant-eyring

Calculate rate constant using Eyring equation (transition state theory) k = (kB·T/h)·exp(-ΔG‡/(R·T))

Arguments:

- delta-h-activation: Enthalpy of activation (kJ/mol)
- delta-s-activation: Entropy of activation (J/(mol·K))
- temp: Temperature (K, default: 298.15)

Returns: Dictionary with rate constant value and unit (s^{-1})

Example:

```typst
#let k = calc-rate-constant-eyring(60, -50, temp: 298.15)
```

Parameters

```
calc-rate-constant-eyring(
  delta-h-activation,
  delta-s-activation,
  temp
)
```

calc-reaction-enthalpy

Calculate the enthalpy change of a reaction using Hess's Law

Arguments:

- reactants: Array of tuples (formula, coefficient), e.g., (("CH4", 1), ("O2", 2))
- products: Array of tuples (formula, coefficient), e.g., (("CO2", 1), ("H2O", 2))
- data: Optional custom thermodynamic data dictionary (defaults to built-in data)

Returns: Dictionary with keys `value` (number) and `unit` (string)

Example:

```typst
#let result = calc-reaction-enthalpy(
  (("CH4", 1), ("O2", 2)),
```

```
  (("CO2", 1), ("H2O", 2))
)
#result.value // -890.3
```

Parameters

```
calc-reaction-enthalpy(
  reactants,
  products,
  data
) -> dict
```

calc-reaction-entropy

Calculate the entropy change of a reaction

Arguments:

- reactants: Array of tuples (formula, coefficient)
- products: Array of tuples (formula, coefficient)
- data: Optional custom thermodynamic data dictionary

Returns: Dictionary with keys `value` (number) and `unit` (string)

Parameters

```
calc-reaction-entropy(
  reactants,
  products,
  data
) -> dict
```

calculate-reaction

Quick reaction calculation from equation

Arguments:

- reactants: Array of tuples (formula, coefficient)
- products: Array of tuples (formula, coefficient)
- temp: Temperature in Kelvin (default: 298.15 K)
- show-details: Show detailed substance data (default: true)
- precision: Number of decimal places (default: 2)
- scientific: Use scientific notation (default: auto)

Returns: Formatted content with all data

Parameters

```
calculate-reaction(
  reactants,
  products,
  temp,
  show-details,
  precision,
  scientific
) -> content
```

detailed-analysis

Detailed reaction analysis with individual substance data

Arguments:

- reactants: Array of tuples (formula, coefficient)
- products: Array of tuples (formula, coefficient)
- temp: Temperature in Kelvin (default: 298.15 K)
- data: Optional custom thermodynamic data dictionary

Returns: Dictionary with detailed analysis including individual substance data

Parameters

```
detailed-analysis(
  reactants,
  products,
  temp,
  data
) -> dict
```

display-analysis

Display reaction analysis in a formatted table

Arguments:

- analysis: Result from analyze-reaction()
- precision: Number of decimal places (default: 2)
- scientific: Use scientific notation (default: auto)

Returns: Content representing a formatted table

Parameters

```
display-analysis(
  analysis,
  precision,
  scientific
) -> content
```

display-detailed-analysis

Display detailed analysis with all substance data

Arguments:

- analysis: Result from detailed-analysis()
- precision: Number of decimal places (default: 2)
- scientific: Use scientific notation (default: auto)

Returns: Content with complete analysis

Parameters

```
display-detailed-analysis(
  analysis,
  precision,
  scientific
) -> content
```

display-kinetics

Display kinetics analysis results

Parameters

```
display-kinetics(analysis)
```

format-number

Format a number with optional scientific notation

Arguments:

- value: Number to format
- precision: Number of decimal places (default: 2)
- scientific: Use scientific notation (default: auto - uses scientific for very large/small numbers)

Parameters

```
format-number(
  value,
  precision,
  scientific
) -> str
```

format-reaction

Format a chemical reaction equation nicely

Arguments:

- reactants: Array of tuples (formula, coefficient)

- products: Array of tuples (formula, coefficient)

Returns: Content representing the formatted equation

Parameters

```
format-reaction(
  reactants,
  products
) -> content
```

format-result

Format a result with value and unit

Parameters

```
format-result(
  result,
  precision,
  scientific
) -> str
```

get-substance-data

Get thermodynamic data for a specific substance

Arguments:

- formula: Chemical formula as string, e.g., "H2O"
- data: Optional custom thermodynamic data dictionary

Returns: Dictionary with keys `delta_Hf`, `S`, and `delta_Gf`

Parameters

```
get-substance-data(
  formula,
  data
) -> dict
```