SSH Model Refactoring

Class Design Template

Complete specification of classes, methods, and attributes

Including user input flow and data ownership

January 28, 2026

Contents

1 User Input Flow

Complete User Workflow

```
# STEP 1: Create ensemble with physical parameters
ensemble = SSHEnsemble(
    t1=2.0,                    # [USER INPUT]
    t2=1.0,                    # [USER INPUT]
    decayConstant=0.1,         # [USER INPUT]
    drivingAmplitude=0.2,      # [USER INPUT]
    drivingFreq=2/3.01         # [USER INPUT]
)

# STEP 2: Add momentum points
ensemble.add_momentum(np.pi/4)  # [USER INPUT]

# STEP 3: Run simulations
tauAxis = np.linspace(0, 300, 200000)
initialConditions = np.array([-0.5, -0.5, 0])

ensemble.run_all(
    tauAxis=tauAxis,           # [USER INPUT - ONLY TIME]
    initialConditions=initialConditions,
    numT=5,
    steadyStateCutoff=25
)

# STEP 4: Visualize (no more input needed)
viz = SSHVisualizer()
model = ensemble.get_model(np.pi/4)
viz.plot_single_time(model, overplot_fourier=True)
```

1.1 Legend

Method Visibility:

- `MethodName()` – Public method
- `_MethodName()` – Protected method
- `__MethodName()` – Private method

Attribute Types:

- **[USER INPUT]** – User must provide this value
- **[COMPUTED]** – System computes this value
- **[STORED]** – Stored from another class

2 Parameters Module

2.1 Class: SSHParameters

Purpose

Store and validate all physical parameters for a single momentum point

User Input Point

Created by `SSHEnsemble` when user calls `add_momentum()`

2.1.1 Attributes

Public:

- `k: float` – [USER INPUT via SSHEnsemble.add_momentum()]
- `t1: float` – [USER INPUT via SSHEnsemble.__init__()]
- `t2: float` – [USER INPUT via SSHEnsemble.__init__()]
- `decayConstant: float` – [USER INPUT]
- `drivingAmplitude: float` – [USER INPUT]
- `drivingFreq: float` – [USER INPUT]

2.1.2 Methods

Public:

- `__init__(k, t1, t2, decayConstant, drivingAmplitude, drivingFreq)`
  Initializes all parameters
- `validate() -> bool`
  Validates parameter ranges and physical constraints
- `to_dict() -> dict`
  Returns parameters as dictionary
- `__repr__() -> str`
  String representation for debugging

3 Physics Module

3.1 Class: SSHHamiltonian

Purpose

Calculate Hamiltonian and eigensystem

User Input Point

None (uses SSHParameters)

3.1.1 Attributes

Private:

- `__params: SSHParameters` – [STORED from constructor]

Protected:

- `_eigenvalues: np.ndarray` – [COMPUTED, cached]
- `_eigenvectors: np.ndarray` – [COMPUTED, cached]
- `_eigenvalues_cached: bool` – [COMPUTED]

3.1.2 Methods

Public:

- `__init__(params: SSHParameters)`
  Stores parameters
- `hamiltonian(tau: float = 0) -> np.ndarray`
  Returns 2×2 Hamiltonian matrix (time-dependent if tau given)
- `eigenvalues() -> np.ndarray`
  Returns eigenvalues (cached after first call)
- `eigenvectors() -> np.ndarray`
  Returns eigenvectors (cached after first call)

Protected:

- `_compute_eigensystem() -> tuple[np.ndarray, np.ndarray]`
  Computes and caches eigenvalues and eigenvectors
- `_clear_cache() -> None`
  Clears cached eigenvalues/eigenvectors

3.2 Class: SSHDissipator

Purpose

Handle dissipation and decay terms

3.2.1 Attributes

Private:

- `__params: SSHParameters` – [STORED from constructor]

3.2.2 Methods

Public:

- `__init__(params: SSHParameters)`
  Stores parameters
- `lindblad_operators() -> list[np.ndarray]`
  Returns list of Lindblad operators
- `decay_rate() -> float`
  Returns the decay rate (γ_-)

4 Solver Module

4.1 Class: ODESolver (Abstract Base)

Purpose

Define interface for all solvers

User Input Point

Not instantiated directly

4.1.1 Methods

Public (Abstract):

- `solve(tauAxis: np.ndarray, initialConditions: np.ndarray, **kwargs) -> np.ndarray`
  Must be implemented by subclasses

4.2 Class: SingleTimeSolver

Purpose

Solve single-time correlation ODEs

User Input Point

Called by `SSHModel.solve()`

4.2.1 Attributes

Private:

- `__hamiltonian: SSHHamiltonian` – [STORED]
- `__dissipator: SSHDissipator` – [STORED]

4.2.2 Methods

Public:

- `__init__(hamiltonian: SSHHamiltonian, dissipator: SSHDissipator)`
  Stores physics objects
- `solve(tauAxis: np.ndarray, initialConditions: np.ndarray, drivingTerm: Callable = None) -> np.ndarray`
  Returns solution array of shape (3, len(tauAxis))
  [USER INPUT: tauAxis, initialConditions via SSHEnsemble.run_all()]

Protected:

- `_ode_system(tau: float, y: np.ndarray, drivingTerm: Callable) -> np.ndarray`
  Defines the ODE system $dy/d\tau = f(\tau, y)$

4.3 Class: DoubleTimeSolver

Purpose

Solve double-time correlation ODEs

4.3.1 Attributes

Private:

- `__hamiltonian: SSHHamiltonian` – [STORED]
- `__dissipator: SSHDissipator` – [STORED]

4.3.2 Methods

Public:

- `solve(tauAxis: np.ndarray, tAxis: np.ndarray, singleTimeSolution: np.ndarray) -> np.ndarray`
  Returns solution array of shape (3, 3, len(tAxis), len(tauAxis))
  [USER INPUT: tauAxis]
  [COMPUTED: tAxis]
  [STORED: singleTimeSolution from SingleTimeSolver]

Protected:

- `_ode_system(tau: float, y: np.ndarray, t: float, singleTimeSolution: np.ndarray) -> np.ndarray`
  Defines the ODE system for double-time correlations

5 Analysis Module

5.1 Class: FourierAnalyzer

Purpose

All Fourier transform operations

User Input Point

Called by `SSHModel` internally

5.1.1 Attributes

Private:

- `__params: SSHParameters` – [STORED from constructor]

5.1.2 Methods

Public:

- `__init__(params: SSHParameters)`
  Stores parameters
- `calculate_coefficients(solution: np.ndarray, tauAxis: np.ndarray, steadyStateCutoff: float, numPeriods: int = 10, n: int = None) -> np.ndarray`
  Returns Fourier coefficients of shape (3, 2n+1)
  [STORED: solution, tauAxis from solver]
  [USER INPUT: steadyStateCutoff]
- `evaluate_expansion(coefficients: np.ndarray, freq: float = None) -> Callable`
  Returns function that evaluates Fourier expansion at given times
- `frequency_axis(tauAxis: np.ndarray) -> np.ndarray`
  Calculates frequency axis for FFT

Protected:

- `_calculate_max_n(tauAxis: np.ndarray) -> int`
  Determines maximum n based on Nyquist frequency

5.2 Class: CurrentCalculator

Purpose

Calculate current operator in time and frequency domains

User Input Point

Called by `SSHModel.calculate_current()`

5.2.1 Attributes

Private:

- `__params: SSHParameters` – [STORED]
- `__fourier_analyzer: FourierAnalyzer` – [STORED]

5.2.2 Methods

Public:

- `__init__(params: SSHParameters, fourier_analyzer: FourierAnalyzer)`
  Stores parameters and analyzer
- `calculate(correlationData: CorrelationData, steadyStateCutoff: float) -> CurrentData`
  Returns CurrentData object with time and frequency domain results
  [STORED: correlationData from SSHModel]
  [USER INPUT: steadyStateCutoff]

Protected:

- `_calculate_current_coefficients(n: int) -> np.ndarray`
  Analytical current coefficients using Bessel functions
- `_calculate_time_domain(expectationCoeff: np.ndarray, currentCoeff: np.ndarray) -> np.ndarray`
  Computes current in time domain
- `_calculate_frequency_domain(timeDomain: np.ndarray, tauAxis: np.ndarray) -> tuple[np.ndarray, np.ndarray]`
  Computes FFT of current operator

6 Data Module

6.1 Class: CorrelationData

Purpose

Store all correlation function results

User Input Point

Created internally by `SSHModel.solve()`

6.1.1 Attributes

Public:

- `singleTime: np.ndarray` – [STORED from SingleTimeSolver]
  Shape: (3, len(tauAxis))
- `doubleTime: np.ndarray` – [STORED from DoubleTimeSolver]
  Shape: (3, 3, len(tAxis), len(tauAxis))
- `tauAxisSec: np.ndarray` – [USER INPUT via SSHEnsemble.run_all()]
- `tauAxisDim: np.ndarray` – [COMPUTED from tauAxisSec]
- `tAxisSec: np.ndarray` – [COMPUTED from steady state]
- `tAxisDim: np.ndarray` – [COMPUTED from tAxisSec]
- `parameters: SSHParameters` – [STORED]
- `fourierCoefficients: np.ndarray` – [COMPUTED by FourierAnalyzer]
  Shape: (3, 2n+1)

6.1.2 Methods

Public:

- `__init__(singleTime, doubleTime, tauAxisSec, tAxisSec, parameters, fourierCoefficients = None)`
  Stores all data
- `get_single_time(operator_index: int) -> np.ndarray`
  Returns single-time correlation for operator (0=σ_-, 1=σ_+, 2=σ_z)
- `get_double_time(i: int, j: int) -> np.ndarray`
  Returns double-time correlation $\langle\sigma_i(t)\sigma_j(t+\tau)\rangle$

6.2 Class: CurrentData

Purpose

Store current operator results

User Input Point

Created internally by `CurrentCalculator`

6.2.1 Attributes

Public:

- `timeDomain: np.ndarray` – [COMPUTED by CurrentCalculator]
- `freqDomain: np.ndarray` – [COMPUTED by CurrentCalculator]
- `freqAxis: np.ndarray` – [COMPUTED from tauAxis]
- `parameters: SSHParameters` – [STORED]

6.2.2 Methods

Public:

- `__init__(timeDomain, freqDomain, freqAxis, parameters)`
  Stores all data

7 Model Module

7.1 Class: SSHModel

Purpose

Orchestrate all components for a single momentum point

User Input Point

Created by `SSHEnsemble.add_momentum()`

7.1.1 Attributes

Public:

- `params: SSHParameters` – [STORED from constructor]

Protected:

- `_hamiltonian: SSHHamiltonian` – [CREATED in ___init___]
- `_dissipator: SSHDissipator` – [CREATED in ___init___]
- `_fourier_analyzer: FourierAnalyzer` – [CREATED in ___init___]
- `_correlation_data: CorrelationData` – [COMPUTED in solve()]
- `_current_data: CurrentData` – [COMPUTED in calculate_current()]

7.1.2 Methods

Public:

- `__init__(parameters: SSHParameters)`
  Creates physics and analysis objects
- `solve(tauAxis: np.ndarray, initialConditions: np.ndarray, numT: int = 5, steadyStateCutoff: float = 25, drivingTerm: Callable = None) -> None`
  Orchestrates solving single and double-time correlations
  [USER INPUT: tauAxis, initialConditions, numT, steadyStateCutoff]
- `calculate_current(steadyStateCutoff: float) -> None`
  Computes current using CurrentCalculator
  [USER INPUT: steadyStateCutoff]

Properties (Read-Only):

- `correlation_data: CorrelationData` – Access correlation results
- `current_data: CurrentData` – Access current results
- `k: float` – Momentum value (from params.k)

Protected:

- `_determine_t_axis(steadyStateCutoff: float, numT: int) -> np.ndarray`
  Calculates tAxis for steady-state initial conditions

7.2 Class: SSHEnsemble

Purpose

Manage multiple momentum points, main user interface

PRIMARY ENTRY POINT FOR USERS

This is where users begin their interaction with the system

7.2.1 Attributes

Protected:

- `_base_params: dict` – [USER INPUT via __init__]
  {t1, t2, decayConstant, drivingAmplitude, drivingFreq}
- `_models: dict[float, SSHModel]` – [CREATED by add_momentum()]

7.2.2 Methods

Public:

- `__init__(t1: float, t2: float, decayConstant: float,`
  `drivingAmplitude: float, drivingFreq: float)`
  PRIMARY USER INPUT POINT for physical parameters
  Stores base parameters for all momentum points
- `add_momentum(k: float | np.ndarray) -> None`
  USER INPUT POINT for momentum values
  Creates SSHModel for each k
- `run_all(tauAxis: np.ndarray, initialConditions: np.ndarray,`
  `numT: int = 5, steadyStateCutoff: float = 25,`
  `drivingTerm: Callable = None, debug: bool = False) -> None`
  USER INPUT POINT for simulation parameters
  Runs solve() and calculate_current() for all models
- `get_model(k: float) -> SSHModel`
  Returns specific momentum model for detailed access

Properties (Read-Only):

- `models: dict[float, SSHModel]` – Access all models
- `momentums: np.ndarray` – Array of all momentum values
- `tauAxisSec: np.ndarray` – Tau axis (returns from first model)
- `tauAxisDim: np.ndarray` – Dimensionless tau axis
- `tAxisSec: np.ndarray` – T axis
- `tAxisDim: np.ndarray` – Dimensionless t axis
- `freqAxis: np.ndarray` – Frequency axis for Fourier transforms

7.3 Class: TotalCurrentCalculator

Purpose

Aggregate current across all momentum points

User Input Point

Called by visualization or analysis code

7.3.1 Attributes

Private:

- `__ensemble: SSHEnsemble` – [STORED from constructor]

7.3.2 Methods

Public:

- `__init__(ensemble: SSHEnsemble)`
  Stores ensemble reference
- `calculate() -> tuple[np.ndarray, np.ndarray]`
  Returns (total_time_domain, total_freq_domain)
  Sums current across all momentum points

8 Visualization Module

8.1 Class: PlotStyler

Purpose

Common styling and formatting for all plots

User Input Point

None (internal defaults)

8.1.1 Attributes

Public:

- `t_label: str` – [CONSTANT] = r"$t\gamma_-$"
- `tau_label: str` – [CONSTANT] = r"$\tau\gamma_-$"
- `plotting_functions: list[Callable]` – [CONSTANT] = [abs, real, imag]

8.1.2 Methods

Public:

- `__init__()`
  Initializes styling constants
- `format_title(params: SSHParameters, k: float = None) -> str`
  Generates consistent plot title with parameters
- `format_operator_label(operator_index: int) -> str`
  Returns LaTeX label for operator (σ_-, σ_+, σ_z)

8.2 Class: SingleTimeCorrelationPlotter

Purpose

Plot single-time correlations only

User Input Point

User calls via `SSHVisualizer`

8.2.1 Attributes

Private:

- `__styler: PlotStyler` – [STORED from constructor]

8.2.2 Methods

Public:

- `__init__(styler: PlotStyler)`
  Stores styler
- `plot(model: SSHModel, overplot_fourier: bool = False) -> None`
  Creates 3×3 subplot of single-time correlations
  [USER PROVIDES: model (via SSHVisualizer)]

Protected:

- `_create_subplot_grid() -> tuple[Figure, np.ndarray]`
  Creates figure with 3×3 subplots
- `_plot_correlation(ax, data: np.ndarray, plotting_func: Callable) -> None`
  Plots single correlation on given axis

8.3 Class: DoubleTimeCorrelationPlotter

Purpose
Plot double-time correlations only

8.3.1 Attributes

Private:

- `__styler: PlotStyler` – [STORED from constructor]

8.3.2 Methods

Public:

- `plot(model: SSHModel, slice: list[tuple[int]] = None,`
  `num_tau_points: int = None, save_figs: bool = False,`
  `subtract_uncorrelated: bool = False) -> None`
  Creates 3D plots of double-time correlations
  [USER PROVIDES: model, options via SSHVisualizer]

Protected:

- `_create_tau_mask(total_points: int, num_points: int) -> np.ndarray`
  Creates mask for downsampling tau axis
- `_plot_correlation_3d(ax, t_data: np.ndarray, tau_data: np.ndarray,`
  `z_data: np.ndarray) -> None`
  Plots single 3D correlation surface

8.4 Class: CurrentPlotter

Purpose
Plot current operator

8.4.1 Attributes

Private:

- `__styler: PlotStyler` – [STORED from constructor]

8.4.2 Methods

Public:

- `plot_single_momentum(model: SSHModel) -> None`
  Plots current for single k value
  [USER PROVIDES: model via SSHVisualizer]
- `plot_total_current(ensemble: SSHEnsemble) -> None`
  Plots summed current across all k
  [USER PROVIDES: ensemble via SSHVisualizer]

Protected:

- `_plot_time_domain(current_data: CurrentData) -> None`
  Creates time-domain current plot
- `_plot_frequency_domain(current_data: CurrentData) -> None`
  Creates frequency-domain current plot

8.5 Class: SSHVisualizer

Purpose
Unified interface for all visualization (Facade pattern)

Main visualization entry point for users

8.5.1 Attributes

Public:

- `styler: PlotStyler` – [CREATED in ___init___]
- `single_time_plotter: SingleTimeCorrelationPlotter`
  [CREATED in ___init___]
- `double_time_plotter: DoubleTimeCorrelationPlotter`
  [CREATED in ___init___]
- `current_plotter: CurrentPlotter` – [CREATED in ___init___]

8.5.2 Methods

Public:

- `__init__()`
  Creates all plotter objects

- `plot_single_time(model: SSHModel, **kwargs) -> None`
  USER ENTRY POINT for single-time plots
  Delegates to single_time_plotter
- `plot_double_time(model: SSHModel, **kwargs) -> None`
  USER ENTRY POINT for double-time plots
  Delegates to double_time_plotter
- `plot_current(model: SSHModel, **kwargs) -> None`
  USER ENTRY POINT for single k current plots
  Delegates to current_plotter.plot_single_momentum()
- `plot_total_current(ensemble: SSHEnsemble, **kwargs) -> None`
  USER ENTRY POINT for total current plots
  Delegates to current_plotter.plot_total_current()

9 Attribute Ownership Table

Complete Reference of All User Inputs

This table shows every piece of user input, where it's stored, and how often it must be provided.

Attribute	Storage Location	Input Method	Frequency
t1, t2	SSHParameters	SSHEnsemble.___init___()	Once at start
decayConstant	SSHParameters	SSHEnsemble.___init___()	Once at start
drivingAmplitude	SSHParameters	SSHEnsemble.___init___()	Once at start
drivingFreq	SSHParameters	SSHEnsemble.___init___()	Once at start
k	SSHParameters	SSHEnsemble.add_momentum()	Once per k
tauAxis	CorrelationData	SSHEnsemble.run_all()	Once at run
initialConditions	Passed to solve()	SSHEnsemble.run_all()	Once at run
numT	Passed to solve()	SSHEnsemble.run_all()	Once at run
steadyStateCutoff	Passed to solve()	SSHEnsemble.run_all()	Once at run
tAxis	CorrelationData (computed)	Auto-computed internally	Never (auto)
singleTimeSolution	CorrelationData (computed)	Auto-computed internally	Never (auto)
doubleTimeSolution	CorrelationData (computed)	Auto-computed internally	Never (auto)
currentTime	CurrentData (computed)	Auto-computed internally	Never (auto)
currentFreq	CurrentData (computed)	Auto-computed internally	Never (auto)
freqAxis	CurrentData (computed)	Auto-computed internally	Never (auto)
fourierCoefficients	CorrelationData (computed)	Auto-computed internally	Never (auto)

10 Key Design Principles

1. **Single Source of Truth**

   Each piece of user input is provided exactly once:

   - Physical parameters → `SSHEnsemble.__init__()`
   - Momentum points → `SSHEnsemble.add_momentum()`
   - Simulation parameters → `SSHEnsemble.run_all()`

2. **Progressive Enhancement**

   User builds up the system step-by-step:

   (a) Create ensemble
   (b) Add momentums
   (c) Run simulations
   (d) Visualize

3. **No Redundant Storage**

   Attributes stored at the most appropriate level:

   - Shared parameters (t1, t2, etc.) → `SSHParameters`
   - Computed results → Data classes (`CorrelationData`, `CurrentData`)
   - Analysis tools → Analyzer classes

4. **Clean Access**

   Results accessed via properties, never by passing data around:

   - `model.correlation_data.singleTime`
   - `model.current_data.timeDomain`
   - `ensemble.tauAxisDim`

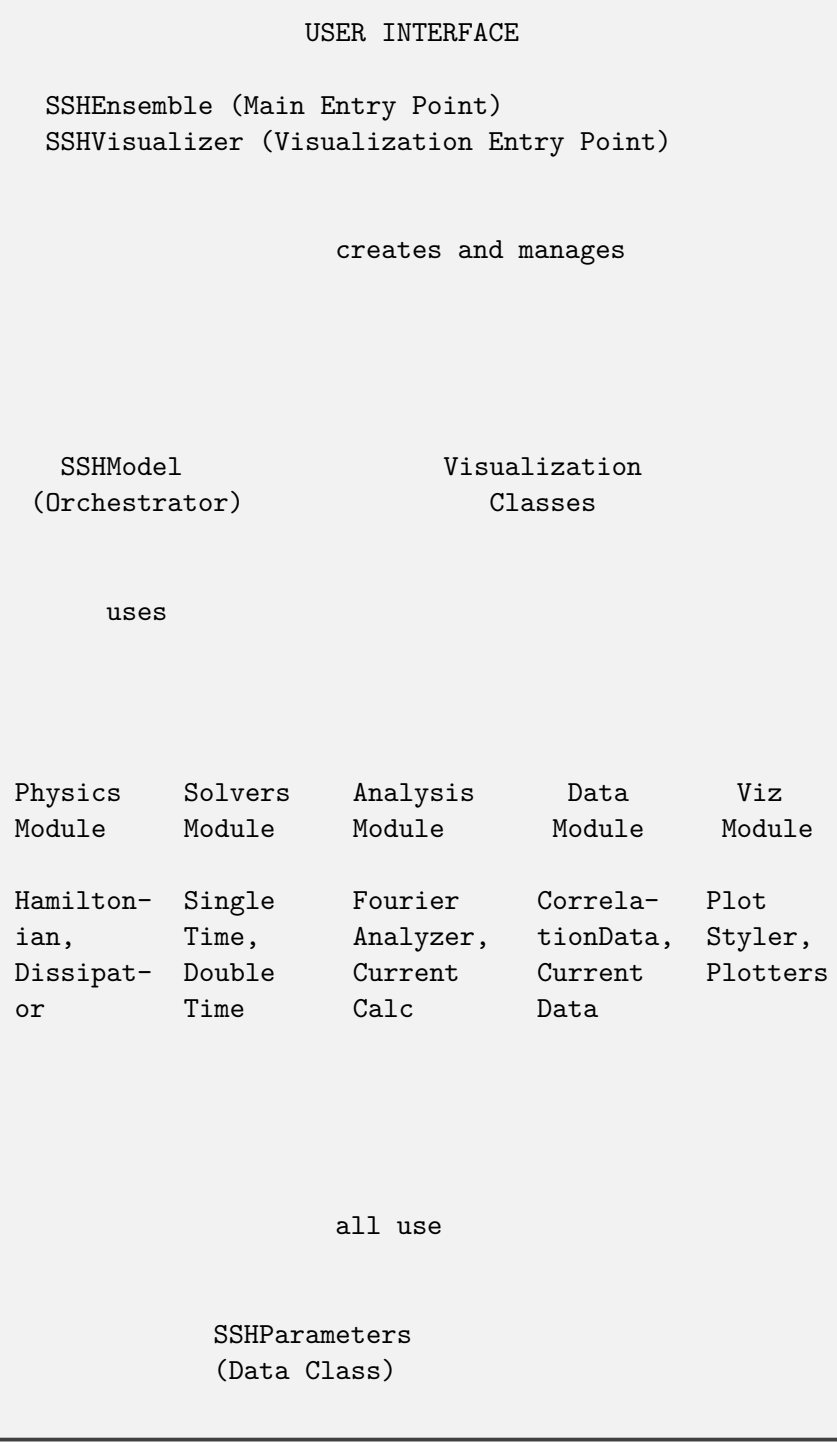
5. **Encapsulation**

   Internal details hidden:

   - User never sees `SingleTimeSolver` directly
   - User never manually creates `FourierAnalyzer`
   - Protected/private methods do the heavy lifting

11 Module Dependencies

Dependency Graph (High to Low Level)



11.1 Key Relationships

- **SSHEnsemble** manages multiple **SSHModel** instances
- **SSHModel** orchestrates all physics, solving, and analysis
- **Physics classes** provide Hamiltonian and dissipation

- **Solver classes** solve ODEs using physics classes
- **Analysis classes** perform Fourier transforms and current calculations
- **Data classes** store results in organized structures
- **Visualization classes** create plots from data classes
- **SSHParameters** is used by all classes that need physical parameters