XFEM Concrete Code Function Summary

Overview

This is an Extended Finite Element Method (XFEM) codebase for modeling crack propagation in reinforced concrete structures. The implementation includes nonlinear concrete behavior (Concrete Damage Plasticity), cohesive zone modeling, and multi-crack analysis capabilities.

---

1. XFEM Analysis Module (`xfem_clean/xfem`)

1.1 Main Analysis Functions

`run_analysis_xfem` (analysis_single.py:39)

Purpose: Main driver for single-crack XFEM displacement-controlled analysis

Parameters: - `model`: XFEMModel configuration object - `nx`, `ny`: Mesh discretization - `nsteps`: Number of load steps - `umax`: Maximum displacement - `law`: Cohesive law (optional) - `return_states`: Whether to return material states

Key Features: - Adaptive substepping with Newton-Raphson solver - Automatic crack initiation based on nonlocal stress averaging - Crack propagation driven by principal tensile stress - Support for linear elastic, Drucker-Prager, and CDP bulk materials - Optional Numba acceleration

Returns: nodes, elements, displacement vector, results array, crack geometry

---

`run_analysis_xfem_multicrack` (xfem/multicrack.py:734)

Purpose: Multi-crack XFEM analysis with Heaviside enrichment

Parameters: - `model`: XFEMModel configuration - `nx`, `ny`: Mesh resolution - `nsteps`: Load steps - `umax`: Maximum displacement - `max_cracks`: Maximum number of cracks allowed - `crack_mode`: "option1" (vertical flexural) or "option2" (flexural + diagonal) - `u_diag_mm`: Displacement threshold for diagonal cracks - `law`: Cohesive law

Key Features: - Multiple concurrent cracks with Heaviside enrichment - Adaptive crack initiation in flexure and shear zones - Continuation/ramping method for enrichment activation - Nonlocal stress averaging for crack nucleation - Gutierrez-style inner equilibrium loop

Returns: nodes, elements, displacement, results, list of cracks

---

1.2 Assembly Functions

`assemble_xfem_system` (xfem/assembly_single.py)

Purpose: Assemble global stiffness matrix and internal force vector

Features: - B-matrix construction with Heaviside and tip enrichment - Subcell integration for cut elements (4x4 subdivision) - Cohesive interface integration (2-point Gauss) - Material nonlinearity support (elastic/DP/CDP) - Numba acceleration option

Returns: K (stiffness), f_int, cohesive state patch, bulk state patch, auxiliary data

---

`assemble_xfem_system_multi` (xfem/multicrack.py:248)

Purpose: Assembly for multiple Heaviside-enriched cracks

Features: - Multi-crack B-matrix with per-crack Heaviside DOFs - Enrichment ramping via `enr_scale` parameter - Cohesive integration per active crack - Rebar contribution via embedded segments

---

1.3 DOF Management

`build_xfem_dofs` (xfem/dofs_single.py)

Purpose: Build DOF mapping for single crack (Heaviside + tip enrichment)

Returns: XFEMDofs object with standard, Heaviside, and tip DOF arrays

---

`build_xfem_dofs_multi` (xfem/multicrack.py:94)

Purpose: Build DOF mapping for multiple Heaviside cracks

Returns: MultiXFEMDofs with standard DOFs and list of Heaviside DOF arrays per crack

---

`transfer_q_between_dofs` (xfem/dofs_single.py)

Purpose: Transfer solution vector when enrichment basis changes

Use Case: After crack initiation or growth, map old DOFs to new DOF numbering

---

1.4 Geometry and Enrichment

`XFEMCrack` class (xfem/geometry.py)

Purpose: Crack geometry representation

Attributes: - `x0, y0`: Crack initiation point - `tip_x, tip_y`: Current crack tip position - `stop_y`: Maximum propagation height - `angle_deg`: Crack angle - `active`: Boolean activation flag

Methods: - `H(x, y)`: Heaviside function (sign of distance to crack) - `cuts_element(xe)`: Check if crack intersects element - `behind_tip(x, y)`: Gating function to deactivate enrichment ahead of tip

---

1.5 Model Configuration

`XFEMModel` class (xfem/model.py:10)

Purpose: Container for all model parameters

Material Parameters: - Concrete: E, nu, ft, fc, Gf - Drucker-Prager: dp_phi_deg, dp_cohesion, dp_H - CDP: cdp_phi_deg, cdp_H, cdp_use_generator - Steel: steel_E, steel_fy, steel_fu, steel_Eh

Solver Parameters: - Newton: newton_maxit, newton_tol_r, newton_beta, line_search - Substepping: max_subdiv - Crack: crack_margin, crack_rho, crack_max_inner

Bulk Material Selector: `bulk_material` = "elastic" | "dp" | "cdp"

---

2. Constitutive Models (`xfem_clean/constitutive.py`)

2.1 Linear Elastic

`LinearElasticPlaneStress` (constitutive.py:158)

Purpose: Plane stress linear elasticity with damage placeholder

Method: - `integrate(mp, eps)`: Returns stress and tangent stiffness

2.2 Drucker-Prager Plasticity

`DruckerPrager` (constitutive.py:286)

Purpose: Associative Drucker-Prager plasticity with plane stress enforcement

Features: - Inscribed yield surface matching Mohr-Coulomb - Isotropic hardening - Return mapping with consistent tangent - Internal Newton iteration for plane stress (szz=0)

Method: - `integrate(mp, eps)`: Returns stress, tangent, updates plastic strain and hardening

---

2.3 Concrete Damage Plasticity (Lite)

`ConcreteCDP` (constitutive.py:393)

Purpose: Simplified CDP with DP plasticity + scalar damage

Features: - Drucker-Prager on effective stress - Split damage variables (tension/compression) - Principal strain-driven damage evolution - Linear softening with fracture energy regularization

Method: - `integrate(mp, eps)`: Returns nominal stress and tangent

2.4 Concrete Damage Plasticity (Lee-Fenves)

`ConcreteCDPReal` (constitutive.py:700)

Purpose: Full Abaqus-like CDP with Lubliner/Lee-Fenves yield surface

Features: - Hyperbolic non-associative plastic potential (dilation angle psi, eccentricity) - Uniaxial hardening/damage tables from cdp_generator - Modified return mapping (freeze flow direction at trial) - Algorithmic tangent consistent with table interpolation

Parameters: - `fb0_fc0`: Biaxial/uniaxial compressive strength ratio - `Kc`: Ratio of second stress invariant on tensile/compressive meridian - Tables: w_tab_m, sig_t_tab_pa, dt_tab (tension) - Tables: eps_in_c_tab, sig_c_tab_pa, dc_tab (compression)

Method: - `integrate(mp, eps)`: Returns nominal stress and tangent with damage applied

---

3. Cohesive Zone Model (`xfem_clean/cohesive_laws.py`)

3.1 Cohesive Law Definitions

`CohesiveLaw` dataclass (cohesive_laws.py:15)

Purpose: Parameters for Mode I cohesive zone

Attributes: - `Kn`: Penalty stiffness [Pa/m] - `ft`: Tensile strength [Pa] - `Gf`: Fracture energy [J/m²] - `law`: "bilinear" or "reinhardt" - `c1, c2`: Reinhardt curve shape parameters

Properties: - `delta0`: Opening at peak stress - `deltaf`: Final opening (complete softening)

`cohesive_update` (cohesive_laws.py:71)

Purpose: Traction-separation law with history

Parameters: - `law`: CohesiveLaw object - `delta`: Current normal opening - `st`: CohesiveState (delta_max, damage) - `visc_damp`: Viscous damping factor

Returns: traction, tangent stiffness, updated state

Logic: - Elastic stage: t = Kn * delta - Softening: bilinear or Reinhardt curve - Unloading: secant stiffness from max opening

---

`cohesive_fracture_energy` (cohesive_laws.py:185)

Purpose: Compute dissipated energy at a cohesive point

Method: Integrates envelope curve from 0 to delta_max

---

4. CDP Parameter Generator (`cdp_generator`)

4.1 Core Calculation

`calculate_stress_strain` (cdp_generator/core.py:31)

Purpose: Generate CDP calibration tables for multiple strain rates

Parameters: - `f_cm` : Mean compressive strength [MPa] - `e_c1` : Strain at peak compression - `e_clim` : Ultimate strain - `l_ch` : Characteristic element length [mm] - `strain_rates` : List of rates [1/s]

Returns: Dictionary with: - Properties: E, G, Gf, ft, dilation angle, Kc, fbfc, l0 - Compression: stress-strain curves, inelastic behavior, damage - Tension: crack opening-stress curves (bilinear/power law), damage

Process: 1. Calculate base material properties (Eurocode) 2. Apply strain rate effects (CEB-FIP) 3. Compute compression Sargin curve 4. Compute tension bilinear/exponential softening 5. Calculate damage variables

---

`calculate_stress_strain_temp` (cdp_generator/core.py:192)

Purpose: Generate CDP tables for multiple temperatures

Temperature Effects: Per Eurocode 2 fire design

4.2 Material Property Functions

`calculate_concrete_strength_properties` (cdp_generator/material_properties.py)

Purpose: Compute f_ck, f_ctm from f_cm per Eurocode 2

`calculate_elastic_modulus` (cdp_generator/material_properties.py)

Purpose: E_ci, E_c from f_cm

`calculate_cdp_parameters` (cdp_generator/material_properties.py)

Purpose: Compute dilation angle, Kc, fb/fc for CDP yield surface

---

`calculate_fracture_energy` (cdp_generator/material_properties.py)

Purpose: Mode I fracture energy G_F from f_cm (Bazant formula)

`calculate_characteristic_length` (cdp_generator/material_properties.py)

Purpose: Compute l_0 = E * G_F / f_t^2

4.3 Compression Behavior

`calculate_compression_behavior` (cdp_generator/compression.py)

Purpose: Sargin compression curve (Eurocode parabola-rectangle)

`calculate_inelastic_compression` (cdp_generator/compression.py)

Purpose: Extract inelastic strain (total minus elastic)

`calculate_compression_damage` (cdp_generator/compression.py)

Purpose: Damage variable d_c from stress degradation

4.4 Tension Behavior

`calculate_tension_bilinear` (cdp_generator/tension.py)

Purpose: Bilinear tension softening (crack opening vs stress)

`calculate_tension_power_law` (cdp_generator/tension.py)

Purpose: Exponential softening curve (Hordijk/Reinhardt)

`calculate_tension_damage` (cdp_generator/tension.py)

Purpose: Damage variable d_t from stress degradation

4.5 Rate and Temperature Effects

`apply_strain_rate_effects` (cdp_generator/strain_rate.py)

Purpose: DIF (Dynamic Increase Factor) per CEB-FIP Model Code

Effects: Increases f_cm, f_ctm, E based on log(strain_rate)

`apply_temperature_effects` (cdp_generator/temperature.py)

Purpose: Strength/stiffness reduction at elevated temperature (Eurocode 2)

4.6 Export Functions

`export_to_abaqus` (cdp_generator/export.py)

Purpose: Write CDP tables in Abaqus *CONCRETE DAMAGE PLASTICITY format

---

`export_to_json` **(cdp_generator/export.py)**

Purpose: JSON export for custom FE codes

---

5. FEM Utilities (`xfem_clean/fem`)

5.1 Mesh Generation

`structured_quad_mesh` **(fem/mesh.py:6)**

Purpose: Generate structured Q4 mesh

Parameters: L (length), H (height), nx, ny

Returns: nodes (nnode x 2), elements (nelem x 4)

5.2 Q4 Element Functions

`q4_shape` **(fem/q4.py)**

Purpose: Q4 shape functions and derivatives

Returns: N (4,), dN_dxi (4,), dN_deta (4,)

`map_global_to_parent_Q4` **(xfem/q4_utils.py)**

Purpose: Inverse isoparametric mapping (x,y) -> (xi, eta) via Newton

---

5.3 Boundary Conditions

`apply_dirichlet` **(fem/bcs.py)**

Purpose: Impose Dirichlet BCs by condensation

Parameters: K, f, fixed (dict of dof:value), q

Returns: free DOF indices, K_ff, f_f, modified q

---

6. Crack Propagation Criteria (`xfem_clean/crack_criteria.py`)

6.1 Stress Averaging

`nonlocal_bar_stress` **(crack_criteria.py)**

Purpose: Weighted averaging of stress in a circular zone

Parameters: - `gp_pos` : Integration point positions [(x,y), ...] - `gp_sig` : Stresses [(sxx, syy, sxy), ...] - `x0, y0` : Center point - `rho` : Averaging radius - `y_max` : Upper limit for integration points - `r_cut` : Cutoff distance

Returns: Averaged 2D stress tensor [sxx, syy, sxy]

Weighting: Gaussian exp(-(r/rho)^2)

`principal_max_dir` (crack_criteria.py)

Purpose: Compute maximum principal stress and direction

Returns: sigma_1, v_1 (eigenvector)

6.2 Candidate Point Generation

`candidate_points_bottom_edge_midpoints` (crack_criteria.py)

Purpose: Generate crack initiation candidate points along bottom edge

Parameters: - `windows`: List of (x_frac_min, x_frac_max) tuples - `spacing`: Point spacing

---

7. Rebar Model (`xfem_clean/rebar.py`)

7.1 Rebar Integration

`prepare_rebar_segments` (rebar.py)

Purpose: Generate 1D rebar segments at specified cover depth

Returns: List of (node_a, node_b, length) tuples

`rebar_contrib` (rebar.py)

Purpose: Compute rebar force and stiffness contribution

Material Model: Bilinear plasticity with hardening

Parameters: - `steel_A_total`: Total cross-sectional area [m²] - `steel_E`: Young's modulus [Pa] - `steel_fy`: Yield strength [Pa] - `steel_fu`: Ultimate strength [Pa] - `steel_Eh`: Hardening modulus [Pa]

Returns: f_rebar (force vector), K_rebar (stiffness matrix)

---

8. Numba Acceleration (`xfem_clean/numba`)

8.1 Cohesive Kernels

`pack_cohesive_law_params` (numba/kernels_cohesive.py)

Purpose: Pack CohesiveLaw into float array for Numba

`cohesive_update_values_numba` **(numba/kernels_cohesive.py)**

Purpose: JIT-compiled cohesive update (returns scalars, no Python objects)

8.2 Bulk Material Kernels

`pack_bulk_params` **(numba/kernels_bulk.py)**

Purpose: Encode material type and parameters

Returns: (kind, params_array) - kind=0: Python material (no Numba) - kind=1: Elastic - kind=2: Drucker-Prager - kind=3: CDP

`elastic_integrate_plane_stress_numba` **(numba/kernels_bulk.py)**

Purpose: Numba-accelerated elastic stress update

`dp_integrate_plane_stress_numba` **(numba/kernels_bulk.py)**

Purpose: Numba DP return mapping

`cdp_integrate_plane_stress_numba` **(numba/kernels_bulk.py)**

Purpose: Numba CDP integration (simplified split damage model)

8.3 Q4 Integration Kernels

`kernels_q4.py`

Purpose: Numba-optimized Q4 element stiffness assembly

---

9. Post-Processing (`xfem_clean/xfem/post.py`)

9.1 Nodal Averaging

`nodal_average_stress_fields` **(xfem/post.py)**

Purpose: Extrapolate integration point stresses to nodes

Method: Weighted averaging by inverse distance

`nodal_average_state_fields` **(xfem/post.py)**

Purpose: Extrapolate damage, plastic strain, etc. to nodes

---

10. Output and Visualization (`xfem_clean/output`)

10.1 VTK Export

`vtk_export.py`

Purpose: Export mesh, displacement, stress, damage to VTK format

Functions: - Write unstructured grid - Cell and point data arrays - Compatible with ParaView

---

10.2 Energy Tracking

`energy.py`

Purpose: Compute global dissipation energies

Tracked Quantities: - W_plastic: Plastic work - W_damage_t: Tensile damage dissipation - W_damage_c: Compressive crushing - W_cohesive: Fracture energy (cohesive interfaces) - W_diss_total: Total dissipation

---

11. State Management (`xfem_clean/xfem/state_arrays.py`)

11.1 Cohesive State Arrays

`CohesiveStateArrays` (state_arrays.py)

Purpose: Flat array storage for cohesive history (Numba-friendly)

Shape: (n_primary, nelem, ngp, 2) for delta_max and damage

Methods: - `get_state(e, gp, k)`: Retrieve CohesiveState object - `get_values(e, gp, k)`: Retrieve (delta_max, damage) tuple - `copy()`: Deep copy for backtracking

---

`CohesiveStatePatch` (state_arrays.py)

Purpose: Sparse update during Newton iterations (avoid full dict copy)

Methods: - `add(k, e, gp, state)`: Add trial state - `apply_to(arrays)`: Apply patch to committed arrays

---

11.2 Bulk State Arrays

`BulkStateArrays` (state_arrays.py)

Purpose: Integration point history for plasticity/damage

Fields: - eps, sigma, eps_p (plane stress) - damage_t, damage_c (scalar damage) - kappa (hardening), w_plastic, w_fract_t, w_fract_c (dissipation) - eps_p6, eps_zz (3D state for plane stress iteration)

Methods: - `get_mp(e, ip)`: Construct MaterialPoint object - `copy()`: Backup for substepping

---

`BulkStatePatch` (state_arrays.py)

Purpose: Sparse bulk state updates

---

12. Material Point (`xfem_clean/material_point.py`)

12.1 MaterialPoint Class

`MaterialPoint` (material_point.py)

Purpose: Container for integration point state

Attributes: - `eps`, `sigma`: Current strain/stress (3D Voigt) - `eps_p`: Plastic strain - `damage`, `damage_t`, `damage_c`: Damage variables - `kappa`: Hardening parameter - `w_plastic`, `w_fract_t`, `w_fract_c`: Energy densities [J/m³] - `extra`: Dict for model-specific history (e.g., eps_zz, eps_p6)

Methods: - `copy_shallow()`: Create trial copy for Newton iteration

---

13. Examples and Validation

13.1 Beam Examples

`run_beam_xfem.py`

Purpose: Simple 3-point bending test

`run_gutierrez_beam.py`

Purpose: Reproduce Gutiérrez 2004 thesis benchmark

`test_nonlinear_concrete_validation.py`

Purpose: CDP validation against analytical solutions

---

14. Key Algorithms Summary

14.1 Newton-Raphson with Substepping

1. Target displacement u_target
2. Try full step with Newton solver
3. If failure: bisect step (adaptive substepping)

4. Stack-based implementation (depth-first)

14.2 Crack Initiation

1. Compute nonlocal averaged stress at candidate points
2. Extract maximum principal stress σ_1
3. If $\sigma_1 \geq ft$ * initiation_factor: activate crack
4. Direction: perpendicular to σ_1 (crack normal aligned with tensile stress)

14.3 Crack Propagation

1. Check stress ahead of current tip
2. If $\sigma_1 \geq ft$: extend crack by segment length
3. Update tip position and angle
4. Rebuild DOFs and re-equilibrate (inner loop)

14.4 Modified Return Mapping (CDP)

1. Freeze plastic flow direction at trial stress
2. Solve for plastic multiplier $\Delta\lambda$ (bisection or Newton)
3. Update hardening variables via uniaxial table interpolation
4. Compute consistent tangent accounting for table slopes

14.5 Continuation/Ramping (Multi-crack)

1. After topology change (crack init/growth), solve at $\alpha=0$ (no enrichment)
2. Gradually increase $\alpha \rightarrow 1.0$ in multiple solves
3. Stabilizes Newton iteration when new DOFs appear

---

15. Units Convention

- **Length**: meters [m]
- **Stress**: pascals [Pa]
- **Force**: newtons [N]
- **Energy**: joules [J]
- **Fracture energy**: [J/m²] or [N/m]

Common conversions: - 1 MPa = 1e6 Pa - 1 mm = 1e-3 m - 1 N/mm² = 1 MPa

---

16. References and Background

16.1 XFEM Theory

- Moës, Dolbow, Belytschko (1999): A finite element method for crack growth without remeshing
- Fries & Belytschko (2010): The extended/generalized finite element method: An overview

16.2 Cohesive Zone

- Barenblatt (1962): Mathematical theory of equilibrium cracks
- Reinhardt et al. (1986): Tensile fracture of concrete

16.3 Concrete Damage Plasticity

- Lubliner et al. (1989): Plastic-damage model for concrete
- Lee & Fenves (1998): Plastic-damage model for cyclic loading
- Abaqus Theory Manual: Concrete Damaged Plasticity

16.4 Implementation Reference

- Gutiérrez (2004): Numerical Analysis of Segmental Tunnel Linings (PhD thesis, TU Delft)

Summary Statistics

- **Total Python modules**: ~50
- **Main analysis drivers**: 2 (single-crack, multi-crack)
- **Constitutive models**: 4 (elastic, DP, CDP-lite, CDP-real)
- **Cohesive laws**: 2 (bilinear, Reinhardt)
- **Numba kernels**: 3 bulk + 1 cohesive
- **CDP generator functions**: ~15 (compression, tension, material properties, rate/temp effects)

Document generated on: 2025-12-29 **Repository**: xfem-concrete **Branch**: claude/pdf-code-summary-lHmfG