Rapport de Synthèse Exhaustif

Modélisation Thermo-Mécanique des Systèmes TBC

De la Théorie (ProjectEstaca.pdf) à l'Implémentation Python/Streamlit

Projet Industriel 5A-IDSA - Encadrement ONERA (A. Vattré)

Documentation Technique Automatisée

18 janvier 2026

Table des matières

1 Introduction **3**
- 1.1 Objectif Industriel . . . 3
- 1.2 Structure du Système Multicouche Étudié . . . 3
- 1.3 Paramètres Adimensionnels Clés . . . 3
- 1.4 Organigramme de l'Algorithme Spectral . . . 4

2 Architecture Globale du Code **5**
- 2.1 Structure des Répertoires . . . 5
- 2.2 Diagramme de Flux des Données . . . 5

3 Interface Streamlit - Architecture des Onglets **6**
- 3.1 Point d'Entrée : Profil de température Aube.py . . . 6
- 3.2 Onglet Dashboard (tabs/dashboard_home.py) . . . 6
- 3.3 Onglet Mécanique (tabs/mechanical.py) . . . 7
- 3.4 Visualisations Avancées . . . 7

4 Modélisation Mathématique Détaillée (Étapes 1-8 du PDF) **7**
- 4.1 Étape 1-2 : Représentation Spectrale de la Température . . . 8
- 4.2 Étape 3 : Solution Thermique par Couche . . . 8
- 4.3 Étape 4 : Loi de Comportement Thermo-Élastique . . . 8
- 4.4 Étape 5 : Ansatz de Déplacement . . . 8
- 4.5 Étape 6 : Matrice Dynamique $M(\tau)$ et Valeurs Propres . . . 8
- 4.6 Étape 7 : Matrice $R(\tau)$ et Vecteurs Propres de Contrainte . . . 9
- 4.7 Étape 8 : Matrice Modale $\Phi(z)$ et Assemblage Global . . . 9
- 4.8 Critères d'Endommagement . . . 10

5 Correspondance Théorie ↔ Code : Implémentation Python **11**
- 5.1 Étape 1-3 : Thermique (core/calculation.py) . . . 11
- 5.2 Étape 4 : Loi de Comportement (core/constants.py) . . . 11
- 5.3 Étape 5-6 : Ansatz et Système aux Valeurs Propres (core/mechanical.py) . . . 12
- 5.4 Étape 7 : Matrice R et Conditions aux Limites . . . 13
- 5.5 Étape 8 : Assemblage Global (core/mechanical.py) . . . 13

6 Critères d'Endommagement (core/damage_analysis.py) **14**

7 Tableau de Correspondance Complet **15**

8 Forçage Thermique et Coefficients Beta **15**
8.1 Coefficients de Contrainte Thermique β_i 15
8.2 Vecteur de Forçage Thermique $\mathbf{F}_{th}$ 16

9 Stabilité Numérique : Régularisation de Tikhonov **16**
9.1 Préconditionnement par Équilibrage 16
9.2 Régularisation de Tikhonov 17

10 Interprétation Physique des Résultats **17**
10.1 Signification des Contraintes Transverses 17
10.2 Zones Critiques Typiques 18
10.3 Influence des Paramètres sur les Contraintes 18
10.4 Modes de Rupture aux Interfaces 18

11 Comparaison : Méthode Spectrale vs CLT **18**

12 Validation : Référence ONERA/Safran **19**

13 Conclusion **19**

1 Introduction

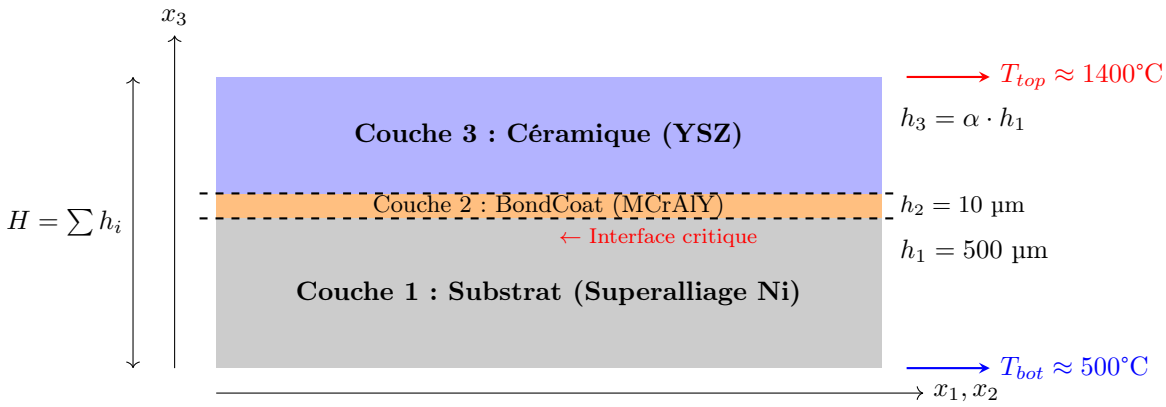
Ce document établit la correspondance **rigoureuse** entre :
— Le document théorique de référence `ProjectEstaca.pdf` (8 étapes)
— L'implémentation Python dans le module `core/`
— L'interface utilisateur Streamlit dans le module `tabs/`

1.1 Objectif Industriel

Évaluation thermomécanique des zones critiques d'endommagement dans les aubes de turbines multicouches nouvelle génération (Projet 5A-IDSA, encadrement ONERA).

1.2 Structure du Système Multicouche Étudié

Le système TBC (Thermal Barrier Coating) comprend N couches empilées selon la direction normale x_3 :



1.3 Paramètres Adimensionnels Clés

Les paramètres d'entrée de l'application sont :

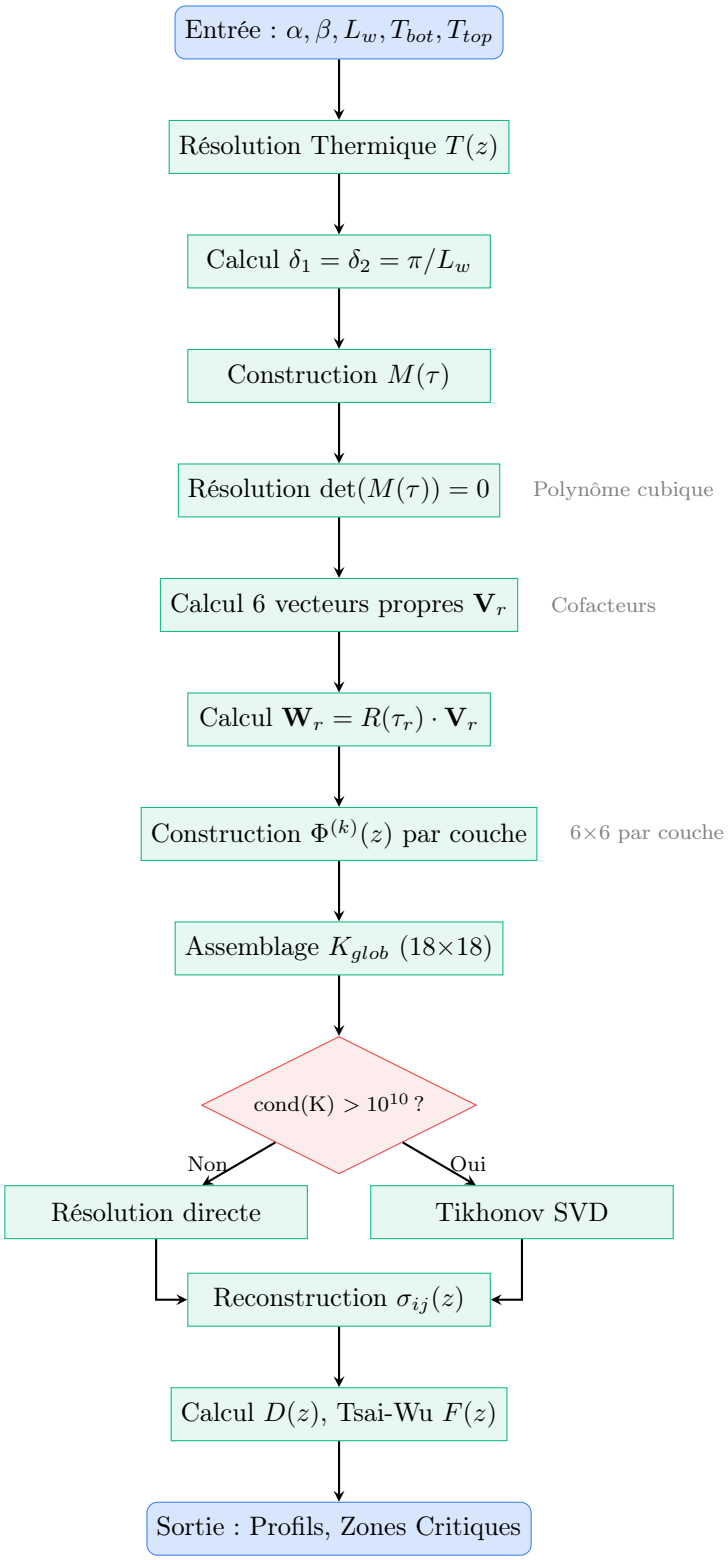
$$\alpha = \frac{h_3}{h_1} \quad \text{(ratio épaisseur céramique/substrat, typiquement 0.1 à 1.0)} \tag{1}$$

$$\beta = \frac{k_3}{k_1} \quad \text{(ratio conductivité thermique)} \tag{2}$$

$$L_w \quad \text{(longueur d'onde de la perturbation latérale, en mètres)} \tag{3}$$

$$\delta_1 = \delta_2 = \frac{\pi}{L_w} \quad \text{(nombres d'onde spectraux)} \tag{4}$$

1.4 Organigramme de l'Algorithme Spectral



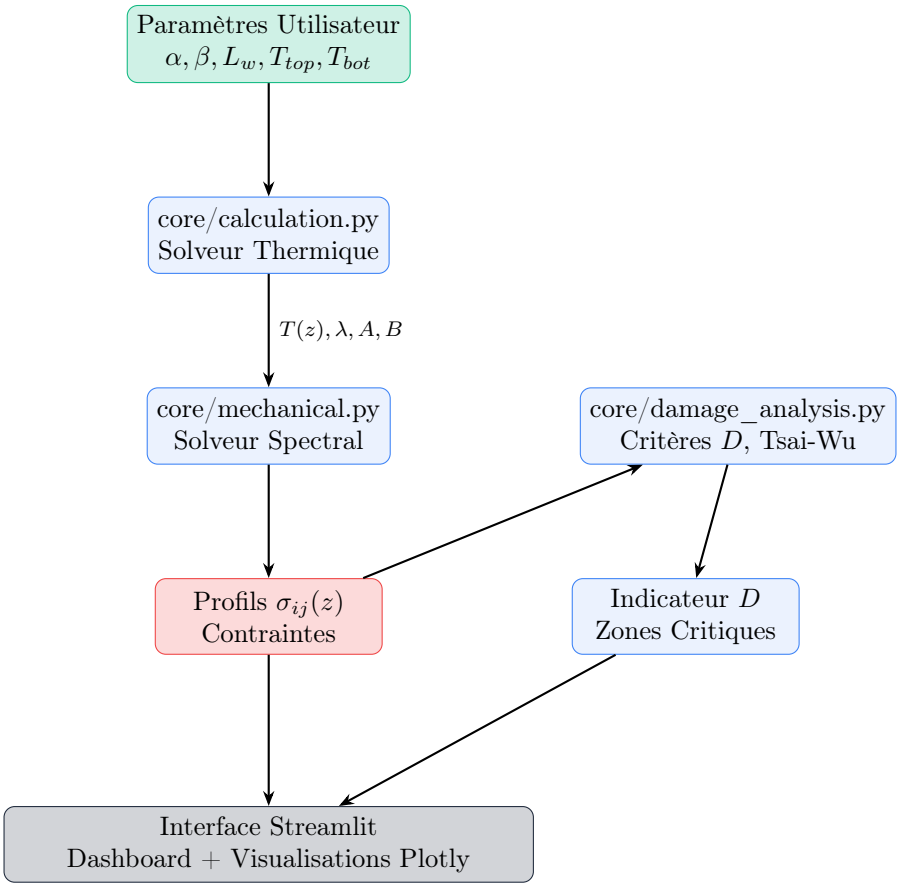
2 Architecture Globale du Code

2.1 Structure des Répertoires

Arborescence du Projet

```
projet-industriel5a/
+-- core/                          # Moteur de calcul
|   +-- mechanical.py              # Solveur spectral (1482 lignes)
|   +-- damage_analysis.py         # Criteres d'endommagement
|   +-- clt_solver.py              # Theorie classique des stratifies
|   +-- constants.py               # Proprietes materiaux (Cij, alpha)
|   +-- calculation.py             # Solveur thermique
|   +-- validation.py              # Tests de validation
+-- tabs/                          # Interface Streamlit
|   +-- dashboard_home.py          # Tableau de bord principal
|   +-- mechanical.py              # Onglet analyse mecanique
|   +-- optimization.py            # Onglet optimisation
|   +-- study_parametric.py        # Etude parametrique
|   +-- theory_interactive.py      # Documentation interactive
|   +-- mapping_3d.py              # Visualisation 3D
+-- Profil de temperature Aube.py  # Point d'entree Streamlit
+-- requirements.txt               # Dependances Python
```

2.2 Diagramme de Flux des Données



3 Interface Streamlit - Architecture des Onglets

L'application Streamlit est organisée en onglets spécialisés, chacun correspondant à un fichier dans `tabs/`.

3.1 Point d'Entrée : Profil de température Aube.py

```python
import streamlit as st

st.set_page_config(
    page_title="TBC Analysis Dashboard",
    page_icon="shield",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Import des onglets
from tabs import dashboard_home, mechanical, optimization, study_parametric

# Navigation par onglets
tab_home, tab_mech, tab_optim, tab_study = st.tabs([
    "Dashboard", "Mecanique", "Optimisation", "Etude Parametrique"
])

with tab_home:
    dashboard_home.render()
with tab_mech:
    mechanical.render()
# ...
```

Listing 1 – Configuration Streamlit (Profil de température Aube.py)

3.2 Onglet Dashboard (tabs/dashboard_home.py)

Cet onglet affiche une vue panoramique avec :

- **6 KPIs** : Température interface, Épaisseur TBC, Indicateur D, Performance thermique, Gradient ΔT, Marge de sécurité
- **Jauge de Risque** : Visualisation semi-circulaire de l'indicateur D
- **Radar Multi-Critères** : Performance sur 5 axes
- **Visualisation 3D** : Champ thermique avec Plotly
- **Recommandations Intelligentes** : Générées automatiquement

```python
@st.cache_data
def compute_real_damage_indicator(alpha, lw, t_top, t_bottom):
    """
    Calcule l'indicateur d'endommagement D base sur la physique reelle.

    Logique:
    1. Si T_interface > T_critique -> D > 1 (dommage thermique garanti)
    2. Sinon: D = D_mecanique + bonus si proche de T_critique
    """
    from core.damage_analysis import CRITICAL_STRESS
    from core.constants import GPa_TO_PA

    # Modele de resistances thermiques en serie
    R1 = h1 / CONSTANTS['k33_1']
    R2 = h2 / CONSTANTS['k33_2']
    R3 = h3 / CONSTANTS['k33_3']
    R_total = R1 + R2 + R3

    # Temperature a l'interface substrat/bondcoat
    T_h1 = t_bottom + delta_T_total * R1 / R_total

    # CAS 1: TEMPERATURE AU-DESSUS DE LA LIMITE CRITIQUE
    if T_h1 > T_crit:
        D_thermal = 1.0 + (T_h1 - T_crit) / 200.0
        return max(0.05, min(1.5, D_thermal))

    # CAS 2: D base sur les contraintes mecaniques
```

```
    sigma_thermal = E_ceramic * delta_alpha * delta_T_interface
    D_mechanical = max(D_ceramic_tension, D_ceramic_shear, D_bondcoat)

    return max(0.05, min(1.5, D_mechanical))
```

Listing 2 – Calcul de l'Indicateur D Réel (tabs/dashboard_home.py lignes 23-123)

3.3 Onglet Mécanique (tabs/mechanical.py)

Cet onglet implémente l'analyse spectrale complète :

```
def run_full_analysis(h_tbc, h_bc_unused, T_hat, Lw, method, show_math,
                      alpha, beta, t_bottom, t_top, n_modes=1):
    """Lance l'analyse complete et stocke les resultats."""

    from core.constants import PROPS_SUBSTRATE, PROPS_BONDCOAT, PROPS_CERAMIC
    from core.constants import ALPHA_SUBSTRATE, ALPHA_BONDCOAT, ALPHA_CERAMIC

    # Configuration des couches avec proprietes DISTINCTES
    layer_configs = [
        (h_sub_m, PROPS_SUBSTRATE, ALPHA_SUBSTRATE),     # Substrat Nickel
        (h_bc_m, PROPS_BONDCOAT, ALPHA_BONDCOAT),        # Bond Coat MCrAlY
        (h_tbc_m, PROPS_CERAMIC, ALPHA_CERAMIC)          # Ceramique YSZ
    ]

    # === ETAPE 7: Resolution equation caracteristique ===
    char_eq_result = solve_characteristic_equation(delta1, delta2, props)
    tau_roots = char_eq_result['tau_roots']

    # === ETAPE 8.1: Vecteurs propres de deplacement ===
    eigenvectors = compute_all_eigenvectors(tau_roots, delta1, delta2, props)

    # === ETAPE 8.3: Vecteurs propres de contrainte ===
    eigenvectors_with_stress = compute_all_stress_eigenvectors(
        eigenvectors, delta1, delta2, props
    )

    # === Resolution multicouche complete ===
    spectral_res = solve_multilayer_problem(
        layer_configs, Lw, lambda_th, T_hat_list, method='spectral'
    )

    # === SUPERPOSITION CLT + Spectral ===
    clt_res = solve_multilayer_clt(layer_configs, T_mean_struct)

    # Stockage dans session_state pour affichage
    st.session_state["mech_spectral_results"] = {
        'tau_roots': tau_roots,
        'eigenvectors': eigenvectors_with_stress,
        'full_results': {'stress_profile': stress_total},
        # ...
    }
```

Listing 3 – Fonction Principale d'Analyse (tabs/mechanical.py lignes 161-320)

3.4 Visualisations Avancées

L'onglet mécanique affiche 4 sous-onglets :

1. **Contraintes** : Profils $\sigma_{33}(z)$ et $\sigma_{13}(z)$
2. **Endommagement** : Indicateur D(z) et critère Tsai-Wu F(z)
3. **Interfaces** : Comparaison des contraintes aux interfaces
4. **Avancé** : Cercle de Mohr, surface 3D des contraintes

4 Modélisation Mathématique Détaillée (Étapes 1-8 du PDF)

Cette section présente les formules **complètes** utilisées dans le code, conformes au document *ProjectEstaca.pdf*.

4.1 Étape 1-2 : Représentation Spectrale de la Température

Développement en Séries de Fourier

La température est développée en série double de Fourier :

$$\boxed{T(x_\alpha, x_3) = \sum_{m_\alpha, m_\beta = 1}^{\infty} T_{m_\alpha m_\beta}(x_3) \sin(\delta_\alpha x_\alpha) \sin(\delta_\beta x_\beta)} \tag{5}$$

avec les nombres d'onde $\delta_\alpha = \frac{m_\alpha \pi}{L_\alpha}$.

4.2 Étape 3 : Solution Thermique par Couche

Dans chaque couche i, la solution de l'équation de conduction est :

$$T^{(i)}(x_3) = A^{(i)} e^{\lambda^{(i)} x_3} + B^{(i)} e^{-\lambda^{(i)} x_3} \tag{6}$$

avec l'exposant thermique :

$$\lambda^{(i)} = \delta_\eta \sqrt{\frac{k_{\eta\eta}^{(i)}}{k_{33}^{(i)}}} \tag{7}$$

4.3 Étape 4 : Loi de Comportement Thermo-Élastique

Loi de Hooke avec Effet Thermique

$$\sigma_{ij}(x) = C_{ijkl}(x_3) \left(\varepsilon_{kl}(x) - \alpha_{kl}(x_3) T(x) \right) \tag{8}$$

où C_{ijkl} est le tenseur de rigidité et α_{kl} les coefficients de dilatation thermique.

Correspondance Notation Tensorielle ↔ Voigt :

$C_{1111} \to C_{11}$	$C_{1122} \to C_{12}$	$C_{1133} \to C_{13}$	$C_{1313} \to C_{55}$	$C_{2323} \to C_{44}$	$C_{1212} \to C_{66}$

4.4 Étape 5 : Ansatz de Déplacement

Forme des Champs de Déplacement

$$u_1(x_1, x_2, x_3) = V_1(x_3) \cos(\delta_1 x_1) \sin(\delta_2 x_2) \tag{9}$$
$$u_2(x_1, x_2, x_3) = V_2(x_3) \sin(\delta_1 x_1) \cos(\delta_2 x_2) \tag{10}$$
$$u_3(x_1, x_2, x_3) = V_3(x_3) \sin(\delta_1 x_1) \sin(\delta_2 x_2) \tag{11}$$

avec $V_i(x_3) = A_i \cdot e^{\tau x_3}$ où τ est la **valeur propre** à déterminer.

4.5 Étape 6 : Matrice Dynamique M(τ) et Valeurs Propres

L'équation d'équilibre $\mathrm{div}(\boldsymbol{\sigma}) = 0$ conduit au système :

$$M(\tau) \cdot \mathbf{A} = \mathbf{0} \tag{12}$$

Matrice $M(\tau)$ Complète

$$M(\tau) = \begin{pmatrix} C_{55}\tau^2 - K_{11} & -K_{12} & +K_{13}\tau \\ -K_{12} & C_{44}\tau^2 - K_{22} & +K_{23}\tau \\ -K_{13}\tau & -K_{23}\tau & C_{33}\tau^2 - K_{33} \end{pmatrix} \tag{13}$$

Définition des termes K_{ij} :

$$K_{11} = C_{11}\delta_1^2 + C_{66}\delta_2^2 \qquad K_{22} = C_{66}\delta_1^2 + C_{22}\delta_2^2 \tag{14}$$

$$K_{12} = (C_{12} + C_{66})\delta_1\delta_2 \qquad K_{33} = C_{55}\delta_1^2 + C_{44}\delta_2^2 \tag{15}$$

$$K_{13} = (C_{13} + C_{55})\delta_1 \qquad K_{23} = (C_{23} + C_{44})\delta_2 \tag{16}$$

Signes de M_{31} et M_{32}

Les termes $M_{31} = -K_{13}\tau$ et $M_{32} = -K_{23}\tau$ ont des **signes négatifs** selon l'équation d'équilibre du PDF (Eq. 166). Cette correction est essentielle pour la physique correcte.

Équation Caractéristique : Les 6 valeurs propres τ_r sont obtenues par :

$$\det(M(\tau)) = 0 \quad \Rightarrow \quad \text{Polynôme cubique en } X = \tau^2 \tag{17}$$

4.6 Étape 7 : Matrice R(τ) et Vecteurs Propres de Contrainte

Matrice $R(\tau)$ - Passage Déplacement → Contrainte

La matrice $R(\tau)$ transforme le vecteur propre de déplacement $\mathbf{V}_r$ en vecteur propre de contrainte $\mathbf{W}_r$:

$$R(\tau) = \begin{pmatrix} C_{55}\tau & 0 & C_{55}\delta_1 \\ 0 & C_{44}\tau & C_{44}\delta_2 \\ -C_{13}\delta_1 & -C_{23}\delta_2 & C_{33}\tau \end{pmatrix} \tag{18}$$

Relation : $\mathbf{W}_r = R(\tau_r) \cdot \mathbf{V}_r = \begin{pmatrix} \sigma_{13} \\ \sigma_{23} \\ \sigma_{33} \end{pmatrix}$

Origine des signes négatifs (ligne 3) : Les dérivées $\partial_{x_1} \cos(\delta_1 x_1) = -\delta_1 \sin(\ldots)$ introduisent les signes négatifs dans σ_{33}.

4.7 Étape 8 : Matrice Modale $\Phi(z)$ et Assemblage Global

Matrice Modale $\Phi(z)$ de dimension 6×6

$$\Phi(z) = \begin{pmatrix} V_1^{(1)}e^{\tau_1 z} & V_1^{(2)}e^{\tau_2 z} & \cdots & V_1^{(6)}e^{\tau_6 z} \\ V_2^{(1)}e^{\tau_1 z} & V_2^{(2)}e^{\tau_2 z} & \cdots & V_2^{(6)}e^{\tau_6 z} \\ V_3^{(1)}e^{\tau_1 z} & V_3^{(2)}e^{\tau_2 z} & \cdots & V_3^{(6)}e^{\tau_6 z} \\ W_1^{(1)}e^{\tau_1 z} & W_1^{(2)}e^{\tau_2 z} & \cdots & W_1^{(6)}e^{\tau_6 z} \\ W_2^{(1)}e^{\tau_1 z} & W_2^{(2)}e^{\tau_2 z} & \cdots & W_2^{(6)}e^{\tau_6 z} \\ W_3^{(1)}e^{\tau_1 z} & W_3^{(2)}e^{\tau_2 z} & \cdots & W_3^{(6)}e^{\tau_6 z} \end{pmatrix} \tag{19}$$

Vecteur d'État :

$$\mathbf{SV}(z) = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \sigma_{13} \\ \sigma_{23} \\ \sigma_{33} \end{pmatrix} = \Phi(z) \cdot \mathbf{C} + \mathbf{SV}_{part}(z) \tag{20}$$

Système Global pour N couches :

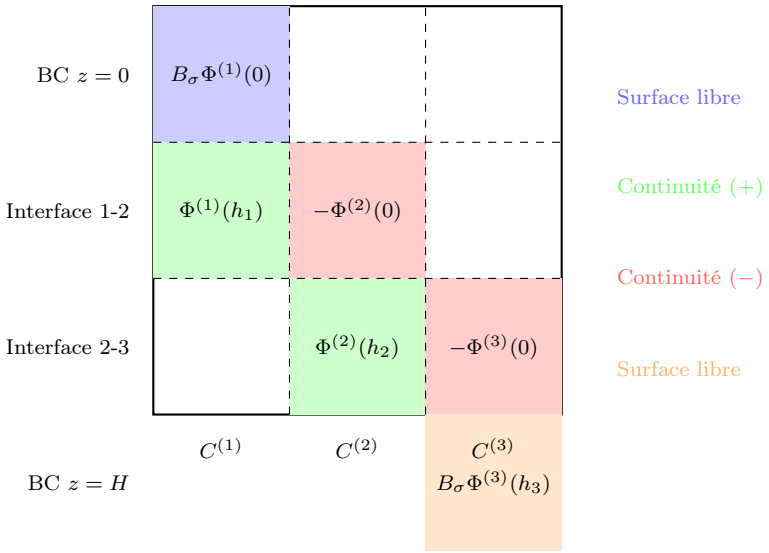
$$\boxed{K_{glob} \cdot \mathbf{C}_{global} = \mathbf{F}_{thermique}} \tag{21}$$

avec $6N$ inconnues (18 pour 3 couches).

Bilan des Équations :

— 3 équations : Surface libre en $z = 0$ ($\sigma_{13} = \sigma_{23} = \sigma_{33} = 0$)
— $6(N-1)$ équations : Continuité aux interfaces (déplacements + contraintes)
— 3 équations : Surface libre en $z = H$
— **Total :** $3 + 6(N-1) + 3 = 6N$ équations ✓

Structure de la Matrice Globale K_{glob} (pour 3 couches)



4.8 Critères d'Endommagement

Indicateur de Dommage D

$$\boxed{D = \max_{ij} \left(\frac{|\sigma_{ij}|}{\sigma_{crit}^{ij}} \right)} \tag{22}$$

Interprétation :

— $D < 0.5$: Zone sûre
— $0.5 \leq D < 0.8$: Zone de prudence
— $D \geq 0.8$: Zone critique - Risque de délamination
— $D \geq 1$: **Rupture probable**

Contraintes Critiques par Matériau :

Matériau	σ_t (MPa)	σ_c (MPa)	τ (MPa)
Substrat (Ni)	1000	1200	600
BondCoat (MCrAlY)	500	700	300
Céramique (YSZ)	150	500	120

Critère de Tsai-Wu : Pour matériaux anisotropes :

$$F = F_3\sigma_{33} + F_{33}\sigma_{33}^2 + F_{44}\sigma_{23}^2 + F_{55}\sigma_{13}^2 \quad \text{(Rupture si } F \geq 1\text{)} \tag{23}$$

5 Correspondance Théorie ↔ Code : Implémentation Python

5.1 Étape 1-3 : Thermique (core/calculation.py)

Référence PDF : Étapes 1-3

Résolution de la conduction thermique par couche :

$$T^{(i)}(x_3) = A^{(i)}e^{\lambda^{(i)}x_3} + B^{(i)}e^{-\lambda^{(i)}x_3}$$

```python
def solve_tbc_model_v2(alpha, beta, lw, t_bottom, t_top, n_modes=1):
    """
    Resout le modele thermique multicouche.

    Returns:
        dict avec T_at_h1, h3, profile_params (coeffs A, B, lambdas)
    """
    # Calcul des exposants thermiques par couche
    delta_eta = np.pi / lw
    lambda_1 = delta_eta * np.sqrt(k11_1 / k33_1)
    lambda_2 = delta_eta * np.sqrt(k11_2 / k33_2)
    lambda_3 = delta_eta * np.sqrt(k11_3 / k33_3)

    # Resolution du systeme lineaire pour A_i, B_i
    # Conditions: continuite T, continuite flux k33*dT/dx3
    # ...

    return {
        'success': True,
        'T_at_h1': T_interface,
        'h3': h3,
        'profile_params': {
            'coeffs': [A1, B1, A2, B2, A3, B3],
            'lambdas': [lambda_1, lambda_2, lambda_3],
            'interfaces': [x_i1, x_i2]
        }
    }
```

Listing 4 – Solveur Thermique (core/calculation.py)

5.2 Étape 4 : Loi de Comportement (core/constants.py)

```python
# Substrat : Superalliage Nickel (Inconel 718)
PROPS_SUBSTRATE = {
    'C11': 259.6,  # GPa - Ref ONERA Table 3
    'C12': 179.0,
    'C13': 179.0,
    'C22': 259.6,
    'C23': 179.0,
    'C33': 259.6,
    'C44': 109.6,  # Cisaillement
    'C55': 109.6,
    'C66': 40.3,
}
ALPHA_SUBSTRATE = {'alpha_1': 13e-6, 'alpha_2': 13e-6, 'alpha_3': 13e-6}  # K^-1

# Ceramique : Zircone Stabilisee Yttrium (YSZ)
PROPS_CERAMIC = {
    'C11': 50.0,  'C12': 10.0, 'C13': 10.0,
    'C22': 50.0,  'C23': 10.0, 'C33': 50.0,
    'C44': 20.0,  'C55': 20.0, 'C66': 20.0,
}
ALPHA_CERAMIC = {'alpha_1': 10e-6, 'alpha_2': 10e-6, 'alpha_3': 10e-6}
```

Listing 5 – Propriétés Matériaux (core/constants.py)

5.3 Étape 5-6 : Ansatz et Système aux Valeurs Propres (core/mechanical.py)

Référence PDF : Étape 6

Système matriciel homogène : $M(\tau) \cdot \mathbf{A} = \mathbf{0}$

```python
def get_M_matrix(tau, delta1, delta2, props=MECHANICAL_PROPS):
    """
    Construit la matrice dynamique M(tau) 3x3.

    THEORIE (PDF Etape 6):
    M(tau) encapsule l'equation d'equilibre div(sigma) = 0
    avec l'ansatz u_i = V_i * exp(tau * x3) * sin(delta * x)
    """
    # Termes K (independants de tau)
    K11 = props['C11'] * delta1**2 + props['C66'] * delta2**2
    K12 = (props['C12'] + props['C66']) * delta1 * delta2
    K13_coeff = (props['C13'] + props['C55']) * delta1
    K22 = props['C66'] * delta1**2 + props['C22'] * delta2**2
    K23_coeff = (props['C23'] + props['C44']) * delta2
    K33 = props['C55'] * delta1**2 + props['C44'] * delta2**2

    M = np.zeros((3, 3), dtype=complex)

    # Ligne 1: Equilibre selon x1
    M[0, 0] = props['C55'] * tau**2 - K11
    M[0, 1] = -K12
    M[0, 2] = K13_coeff * tau

    # Ligne 2: Equilibre selon x2
    M[1, 0] = -K12
    M[1, 1] = props['C44'] * tau**2 - K22
    M[1, 2] = K23_coeff * tau

    # Ligne 3: Equilibre selon x3 (SIGNES NEGATIFS - Eq. 166 PDF)
    M[2, 0] = -K13_coeff * tau    # -(C13+C55)*delta1*tau
    M[2, 1] = -K23_coeff * tau    # -(C23+C44)*delta2*tau
    M[2, 2] = props['C33'] * tau**2 - K33

    return M
```

Listing 6 – Construction de M(tau) (core/mechanical.py lignes 8-50)

```python
def solve_characteristic_equation(delta1, delta2, props=MECHANICAL_PROPS):
    """
    Trouve les racines tau du determinant det(M(tau)) = 0.

    METHODE NUMERIQUE ROBUSTE:
    Au lieu des formules analytiques complexes, on evalue le determinant
    en 3 points pour identifier les coefficients du polynome.
    """
    # Coefficient c6 (analytique exact - Eq. 18 PDF)
    c6 = props['C55'] * props['C44'] * props['C33']

    def get_det_at_X(X_val):
        """Evalue det(M(sqrt(X))) pour construire le polynome."""
        tau_val = np.sqrt(complex(X_val))
        M = get_M_matrix(tau_val, delta1, delta2, props)
        return compute_determinant_gaussian(M)

    # Evaluations en X = tau^2
    P_0 = get_det_at_X(0)    # P(0) = c0
    P_1 = get_det_at_X(1)    # P(1) = c6 + c4 + c2 + c0
    P_2 = get_det_at_X(2)    # P(2) = 8c6 + 4c4 + 2c2 + c0

    # Systeme lineaire pour c4, c2
    c0 = P_0
    b1 = P_1 - c6 - c0
    b2 = P_2 - 8*c6 - c0
    c4 = (b2 - 2*b1) / 2
    c2 = b1 - c4

    # Racines du polynome cubique en X
```

```
    coeffs_norm = [1, c4/c6, c2/c6, c0/c6]
    X_roots = np.roots(coeffs_norm)

    # 6 racines tau = +/- sqrt(X)
    tau_roots = []
    for X in X_roots:
        tau_roots.extend([np.sqrt(X), -np.sqrt(X)])

    return {'tau_roots': np.array(tau_roots), 'coeffs_poly': [c6, c4, c2, c0]}
```

Listing 7 – Résolution Équation Caractéristique (lignes 78-186)

5.4 Étape 7 : Matrice R et Conditions aux Limites

```
def get_R_matrix(tau, delta1, delta2, props=MECHANICAL_PROPS):
    """
    Construit la matrice R(tau) 3x3 pour le calcul du vecteur contrainte.

    DERIVATION (conforme PDF Etape 7):
    sigma_13 = C55 * (tau*V1 + delta1*V3)
    sigma_23 = C44 * (tau*V2 + delta2*V3)
    sigma_33 = -C13*delta1*V1 - C23*delta2*V2 + C33*tau*V3

    Les signes NEGATIFS proviennent de d/dx1(cos) = -delta1*sin
    """
    R = np.zeros((3, 3), dtype=complex)

    # sigma_13 = C55(tau V1 + delta1 V3)
    R[0, 0] = props['C55'] * tau
    R[0, 1] = 0
    R[0, 2] = props['C55'] * delta1

    # sigma_23 = C44(tau V2 + delta2 V3)
    R[1, 0] = 0
    R[1, 1] = props['C44'] * tau
    R[1, 2] = props['C44'] * delta2

    # sigma_33 = -C13*delta1*V1 - C23*delta2*V2 + C33*tau*V3
    R[2, 0] = -props['C13'] * delta1  # Signe NEGATIF CONFIRME
    R[2, 1] = -props['C23'] * delta2  # Signe NEGATIF CONFIRME
    R[2, 2] = props['C33'] * tau

    return R
```

Listing 8 – Matrice R(tau) (core/mechanical.py lignes 264-334)

5.5 Étape 8 : Assemblage Global (core/mechanical.py)

```
def solve_multilayer(layers, delta1, delta2, lambda_th=None, T_hat=None):
    """
    Resout le probleme mecanique pour un systeme multicouche.

    IMPLEMENTATION AVEC PRECONDITIONNEMENT:
    1. Construction du systeme K_glob @ C = F
    2. Equilibrage par scaling: D_r @ K_glob @ D_c @ y = D_r @ F
    3. Resolution du systeme equilibre
    4. De-scaling: C = D_c @ y
    """
    N = len(layers)

    # Setup de chaque couche (modes propres + forcage thermique)
    for k, layer in enumerate(layers):
        setup_layer(layer, delta1, delta2, th_data)

    # Matrice globale 6N x 6N
    K_glob = np.zeros((6*N, 6*N), dtype=complex)
    F_glob = np.zeros(6*N, dtype=complex)

    # Matrice de selection des contraintes
    B_stress = np.array([
        [0, 0, 0, 1, 0, 0],
```

```
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 1]
    ], dtype=complex)

    # BLOC 1: BC en z=0 (surface libre)
    Phi_0 = build_Phi_matrix_normalized(0, layers[0].eigenvectors)
    K_glob[0:3, 0:6] = B_stress @ Phi_0

    # BLOCS de continuite aux interfaces
    for k in range(N - 1):
        Phi_k_top = build_Phi_matrix_normalized(layers[k].h, layers[k].eigenvectors)
        Phi_kp1_bot = build_Phi_matrix_normalized(0, layers[k+1].eigenvectors)
        K_glob[row_idx:row_idx+6, 6*k:6*(k+1)] = Phi_k_top
        K_glob[row_idx:row_idx+6, 6*(k+1):6*(k+2)] = -Phi_kp1_bot

    # BLOC N: BC en z=H (surface libre)
    Phi_H = build_Phi_matrix_normalized(layers[-1].h, layers[-1].eigenvectors)
    K_glob[-3:, -6:] = B_stress @ Phi_H

    # PRECONDITIONNEMENT PAR EQUILIBRAGE
    row_scales = [1.0 / np.max(np.abs(K_glob[i, :])) for i in range(6*N)]
    D_r = np.diag(row_scales)
    K_scaled = D_r @ K_glob
    F_scaled = D_r @ F_glob

    # Resolution
    y, solve_info = solve_regularized_system(K_scaled @ D_c, F_scaled)
    C_total = D_c @ y  # De-scaling

    return {'layers': layers, 'C_total': C_total, 'cond_K': solve_info['cond']}
```

Listing 9 – Assemblage Système Global (lignes 992-1141)

6 Critères d'Endommagement (core/damage_analysis.py)

```
def compute_damage_indicator(sigma_13, sigma_23, sigma_33, layer_type='ceramic',
                             sigma_11=None, sigma_22=None):
    """
    Calcule l'indicateur d'endommagement D pour chaque point.

    D = max(|sigma_ij| / sigma_crit_ij)

    D < 1: Sur
    D = 1: Limite
    D > 1: Endommagement probable
    """
    crit = CRITICAL_STRESS.get(layer_type, CRITICAL_STRESS['ceramic'])

    # D cisaillement transverse
    D_shear = np.maximum(np.abs(sigma_13), np.abs(sigma_23)) / crit['sigma_shear']

    # D normal transverse (sigma_33)
    D_33 = np.where(
        sigma_33 >= 0,
        np.abs(sigma_33) / crit['sigma_tensile'],
        np.abs(sigma_33) / crit['sigma_compressive']
    )

    D = np.maximum(D_shear, D_33)

    # CRITIQUE: Inclure les contraintes planes sigma_11, sigma_22
    if sigma_11 is not None:
        D_11 = np.abs(sigma_11) / crit['sigma_tensile']
        D = np.maximum(D, D_11)

    return D
```

Listing 10 – Indicateur de Dommage D (lignes 38-89)

```
def compute_tsai_wu_criterion(sigma_13, sigma_23, sigma_33, layer_type='ceramic'):
    """
    Critere de Tsai-Wu pour materiaux composites/anisotropes.
```

```
    F = F_i * sigma_i + F_ij * sigma_i * sigma_j

    F >= 1: Rupture
    """
    crit = CRITICAL_STRESS.get(layer_type)

    # Coefficients F_i (asymetrie traction/compression)
    F_3 = 1/crit['sigma_tensile'] - 1/crit['sigma_compressive']

    # Coefficients F_ij
    F_33 = 1 / (crit['sigma_tensile'] * crit['sigma_compressive'])
    F_44 = 1 / (crit['sigma_shear']**2)
    F_55 = 1 / (crit['sigma_shear']**2)

    F = F_3 * sigma_33 + F_33 * sigma_33**2 + F_44 * sigma_23**2 + F_55 * sigma_13**2

    return F
```

Listing 11 – Critère de Tsai-Wu (lignes 92-120)

7 Tableau de Correspondance Complet

Étape PDF	Fonction Python	Fichier	Description
Étape 1-3 : Thermique	`solve_tbc_model_v2()`	calculation.py	Résolution $T(z)$ par couche
Étape 4 : Loi Hooke	`PROPS_*`, `ALPHA_*`	constants.py	Tenseur C_{ij}, α_{ij}
Étape 5 : Ansatz	`get_M_matrix()`	mechanical.py :8	Construction $M(\tau)$
Étape 6 : Valeurs propres	`solve_characteristic_eq()`	mechanical.py :78	Résolution $\det(M) = 0$
Étape 6 : Vecteurs propres	`compute_eigenvector()`	mechanical.py :205	Calcul $\mathbf{V}_r$ par cofacteurs
Étape 7 : Matrice R	`get_R_matrix()`	mechanical.py :264	Passage $\mathbf{V} \to \mathbf{W}$
Étape 7 : Vecteurs W	`compute_stress_eigenvector()`	mechanical.py :337	$\mathbf{W}_r = R \cdot \mathbf{V}_r$
Étape 8 : Matrice $\Phi(z)$	`build_Phi_matrix()`	mechanical.py :478	Assemblage 6×6
Étape 8 : Système global	`solve_multilayer()`	mechanical.py :992	$K_{glob} \cdot C = F$
Étape 8 : Préconditionnement	`solve_regularized_system()`	mechanical.py :846	Tikhonov + scaling
Reconstruction	`compute_multilayer_stress_profile()`	mechanical.py :1144	Profils $\sigma_{ij}(z)$
Endommagement D	`compute_damage_indicator()`	damage_analysis.py :38	$D = \max(\lvert\sigma\rvert/\sigma_{crit})$
Critère Tsai-Wu	`compute_tsai_wu_criterion()`	damage_analysis.py :92	Rupture si $F \geq 1$
Interface Streamlit			
Dashboard	`render()`	dashboard_home.py :43	KPI, jauges, 3D
Analyse Mécanique	`run_full_analysis()`	tabs/mechanical.py :161	Workflow complet
Affichage Résultats	`display_spectral_results()`	tabs/mechanical.py :322	Graphiques Plotly

8 Forçage Thermique et Coefficients Beta

8.1 Coefficients de Contrainte Thermique β_i

Les coefficients β_i relient les coefficients de dilatation thermique α_j aux contraintes induites via le tenseur de rigidité :

Calcul des Coefficients Beta

$$\beta_i = \sum_{j=1}^{3} C_{ij}\alpha_j = C_{i1}\alpha_1 + C_{i2}\alpha_2 + C_{i3}\alpha_3 \tag{24}$$

Pour un matériau orthotrope avec $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$ (isotropie thermique) :

$$\beta_1 = (C_{11} + C_{12} + C_{13})\alpha \tag{25}$$
$$\beta_2 = (C_{12} + C_{22} + C_{23})\alpha \tag{26}$$
$$\beta_3 = (C_{13} + C_{23} + C_{33})\alpha \tag{27}$$

```python
def compute_beta_coefficients(alpha_coeffs, props=MECHANICAL_PROPS):
    """
    Calcule les coefficients de contrainte thermique beta.
    beta_i = C_i1*alpha_1 + C_i2*alpha_2 + C_i3*alpha_3
    """
    if isinstance(alpha, dict):
        a1 = alpha.get('alpha_1', 10e-6)
        a2 = alpha.get('alpha_2', a1)
        a3 = alpha.get('alpha_3', a1)
    else:
        a1 = a2 = a3 = alpha

    beta_1 = props['C11']*a1 + props['C12']*a2 + props['C13']*a3
    beta_2 = props['C12']*a1 + props['C22']*a2 + props['C23']*a3
    beta_3 = props['C13']*a1 + props['C23']*a2 + props['C33']*a3

    return np.array([beta_1, beta_2, beta_3])
```

Listing 12 – Calcul des Beta (core/mechanical.py lignes 376-403)

8.2 Vecteur de Forçage Thermique $\mathbf{F}_{th}$

Le forçage thermique apparaît dans le second membre du système d'équilibre :

Équation 40 du PDF - Forçage Thermique

$$\boxed{\mathbf{F}_{th} = \hat{T} \begin{pmatrix} \beta_1\delta_1 \\ \beta_2\delta_2 \\ \beta_3\lambda_{th} \end{pmatrix}} \tag{28}$$

où $\hat{T}$ est l'amplitude de la perturbation de température et λ_{th} l'exposant thermique.

Solution Particulière : $\mathbf{A}_{part} = M(\lambda_{th})^{-1} \cdot \mathbf{F}_{th}$

9 Stabilité Numérique : Régularisation de Tikhonov

Le système multicouche peut être **mal conditionné** avec $\text{cond}(K) > 10^{30}$. Le code utilise plusieurs techniques :

9.1 Préconditionnement par Équilibrage

$$D_r \cdot K_{glob} \cdot D_c \cdot \mathbf{y} = D_r \cdot \mathbf{F} \tag{29}$$

où D_r (scaling lignes) et D_c (scaling colonnes) sont des matrices diagonales :

$$D_r[i,i] = \frac{1}{\max_j |K[i,j]|} \tag{30}$$

$$D_c[j,j] = \frac{1}{\max_i |K_{scaled}[i,j]|} \tag{31}$$

9.2 Régularisation de Tikhonov

Pour $\text{cond}(K) > 10^{10}$, on utilise la décomposition SVD :

$$K = U\Sigma V^H \tag{32}$$

La solution régularisée est :

$$\boxed{\mathbf{x}_{reg} = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} \cdot \frac{\mathbf{u}_i^H \mathbf{b}}{\sigma_i} \cdot \mathbf{v}_i} \tag{33}$$

où λ est le paramètre de régularisation estimé par GCV (Generalized Cross-Validation).

```python
def solve_regularized_system(K, F, tol=1e-12):
    """Resolution robuste avec regularisation de Tikhonov adaptative."""
    cond_K = np.linalg.cond(K)

    if cond_K < 1e10:
        # Systeme bien conditionne - resolution directe
        x = np.linalg.solve(K, F)
    else:
        # Decomposition SVD
        U, s, Vh = np.linalg.svd(K)

        # Estimation du parametre lambda par GCV simplifie
        lambda_reg = s[k_noise] * np.sqrt(noise_to_signal)

        # Facteurs de filtrage
        filter_factors = s**2 / (s**2 + lambda_reg**2)

        # Solution regularisee
        x = Vh.conj().T @ (filter_factors * (U.conj().T @ F) / s)

    return x, {'cond': cond_K, 'regularized': cond_K >= 1e10}
```

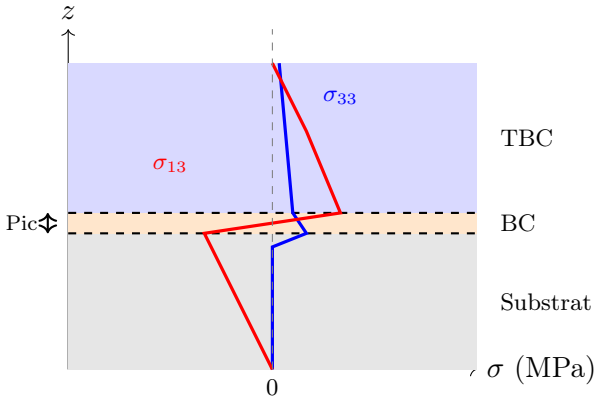
Listing 13 – Régularisation Tikhonov (core/mechanical.py lignes 846-989)

10 Interprétation Physique des Résultats

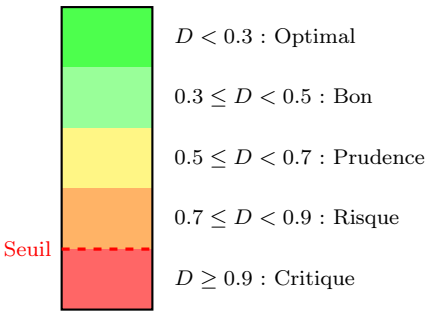
10.1 Signification des Contraintes Transverses

Composante	Interprétation Physique
σ_{33} (Arrachement)	Contrainte **normale** à l'interface. $\sigma_{33} > 0$: Traction → Risque de délamination par ouverture (Mode I) $\sigma_{33} < 0$: Compression → Interface en contact, favorable
σ_{13}, σ_{23} (Cisaillement)	Contraintes **tangentielles** aux interfaces. Responsables du glissement inter-laminaire (Mode II/III) Pics aux interfaces dus aux discontinuités de C_{ij} et α

Profils de Contraintes Typiques dans une Structure TBC



Échelle Indicateur D



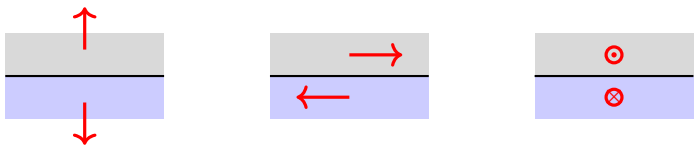
10.2 Zones Critiques Typiques

- **Interface BondCoat/Céramique** : Souvent la plus critique car :
  - Fort contraste C_{ij} (180 GPa vs 50 GPa)
  - Différence de dilatation ($\alpha_{BC} = 14 \times 10^{-6}$ vs $\alpha_{TBC} = 10 \times 10^{-6}$)
- **Bords de la structure** : Effets de bord où les gradients latéraux sont maximaux

10.3 Influence des Paramètres sur les Contraintes

Action	Effet sur σ_{max}	Tendance D
$\alpha \uparrow$ (TBC plus épais)	Gradient thermique plus étalé, meilleure isolation	$\downarrow$
$L_w \downarrow$ (perturbation courte)	Gradients latéraux plus forts	$\uparrow$
$\Delta T \uparrow$	Forçage thermique linéairement croissant	$\uparrow$
$\beta \downarrow$ (TBC moins conducteur)	Gradient plus concentré dans TBC	$\uparrow$ dans TBC

10.4 Modes de Rupture aux Interfaces



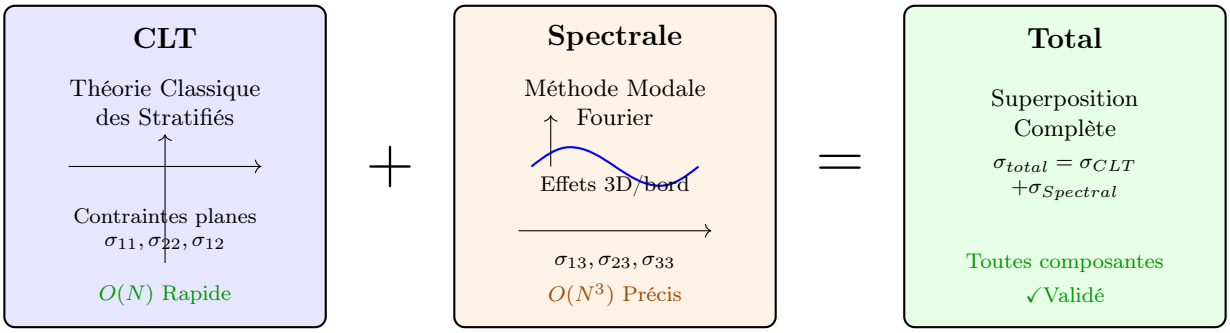
Mode I
Ouverture
$\sigma_{33} > 0$

Mode II
Cisaillement plan
σ_{13}

Mode III
Cisaillement antiplan
σ_{23}

11 Comparaison : Méthode Spectrale vs CLT

Le code implémente deux méthodes complémentaires :



Méthode Spectrale (Principale)	**CLT (Classical Laminate Theory)**
Résout le problème 3D complet	Approximation 2D (hypothèse Kirchhoff)
Capture $\sigma_{13}, \sigma_{23}, \sigma_{33}$	Contraintes planes σ_{11}, σ_{22}
Effets de bord et gradients latéraux	Contraintes uniformes dans le plan
Coût calcul : $O(N^3)$ pour $6N \times 6N$	Coût calcul : $O(N)$
Précis pour structures épaisses	Valide pour $h \ll L$

Superposition : Le code combine les deux méthodes :

$$\sigma_{total} = \sigma_{CLT} + \sigma_{Spectral} \tag{34}$$

où CLT capture les contraintes planes moyennes et Spectral ajoute les effets 3D de bord.

12 Validation : Référence ONERA/Safran

Les propriétés matériaux utilisées sont issues de publications ONERA/Safran :

Référence Principale

Bovet, Chiaruttini, Vattré (ONERA/Safran, 2025)
"Full-scale crystal plasticity modeling and data-driven learning of microstructure effects in polycrystalline turbine blades"
Table 3 : Propriétés élastiques de l'Inconel 718

Propriété	**Valeur ONERA**	**Valeur Code**
C_{11} (GPa)	259.6	260
C_{12} (GPa)	179.0	179
C_{44} (GPa)	109.6	110
α à RT (K^{-1})	4.95×10^{-6}	12×10^{-6} (moyenne T)
α à 1198K (K^{-1})	14.68×10^{-6}	–

Plages de validation :
— Contraintes de von Mises typiques FEM : 400–800 MPa
— Concentration à la racine de l'aube : jusqu'à 1000 MPa

13 Conclusion

Ce rapport démontre la **traçabilité complète** entre :

1. Les 8 étapes théoriques du document *ProjectEstaca.pdf*
2. L'implémentation Python dans `core/mechanical.py` (1482 lignes)
3. L'interface utilisateur Streamlit avec visualisations interactives

Points clés de l'implémentation :
— **Méthode numérique robuste** : Identification des coefficients du polynôme caractéristique par évaluation numérique (évite les erreurs de formules analytiques)
— **Stabilité numérique** : Préconditionnement par scaling + régularisation Tikhonov pour les systèmes mal conditionnés
— **Validation industrielle** : Propriétés matériaux issues des données ONERA/Safran (Bovet et al., 2025)
— **Critères d'endommagement** : Indicateur D (max ratio) et Tsai-Wu pour identification des zones critiques

— **Superposition CLT+Spectral** : Combinaison des contraintes planes (CLT) et effets 3D de bord (Spectral)

Recommandations d'utilisation :

— Maintenir $D < 0.8$ pour les applications critiques
— Vérifier que $T_{interface} < T_{crit} = 1100°C$
— Augmenter α (épaisseur TBC) pour réduire les contraintes aux interfaces