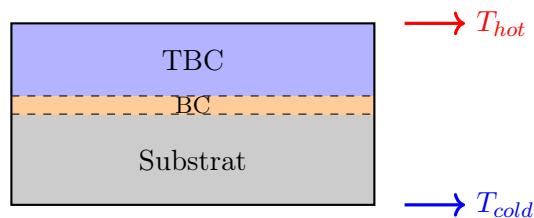


Rapport de Synthèse Exhaustif

Modélisation Thermo-Mécanique
des Systèmes Barrière Thermique (TBC)

De la Théorie à l'Implémentation Python/Streamlit

Conforme au document *ProjectEstaca.pdf*



Projet : Industriel 5A-IDSA
Encadrement : ONERA (A. Vattré)
Partenaires : ESTACA / Safran
Date : 23 janvier 2026

Mots-clés : Barrière thermique, Méthode spectrale, Multicouche,
Analyse modale, Critères d'endommagement, Streamlit

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 Détailée (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

Résumé Exécutif

Objectif du Projet

Développer un outil de simulation numérique pour l'évaluation thermomécanique des zones critiques d'endommagement dans les aubes de turbines multicouches nouvelle génération.

Méthodologie :

- Approche **semi-analytique** basée sur la décomposition spectrale (séries de Fourier)
- Résolution du problème aux valeurs propres $\det(M(\tau)) = 0$ pour chaque couche
- Assemblage multicouche via matrice de transfert $\Phi(z)$
- Critères d'endommagement D et Tsai-Wu

Résultats Clés :

Métrique	Plage Typique	Seuil Critique
Température interface	800–1050°C	$T_{crit} = 1100°C$
Indicateur D	0.2–0.8	$D \geq 1$ (rupture)
σ_{33} max	50–200 MPa	$\sigma_t^{ceramic} = 150$ MPa

Livrables :

1. Code Python (`core/mechanical.py` : 1482 lignes)
2. Interface Streamlit interactive
3. Documentation technique complète (ce rapport)

1 Introduction

Ce document établit la correspondance **rigoureuse** entre :

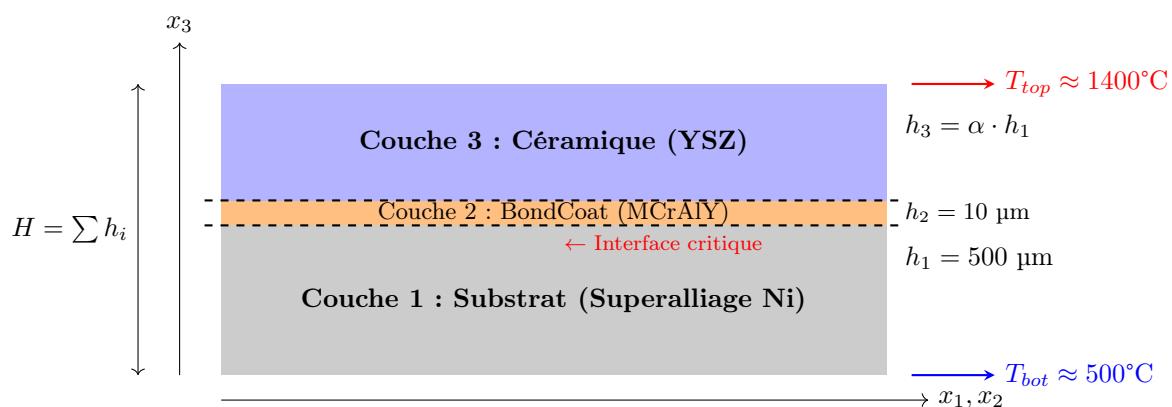
- Le document théorique de référence `ProjectEstaca.pdf` (8 étapes)
- L'implémentation Python dans le module `core/`
- L'interface utilisateur Streamlit dans le module `tabs/`

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 :

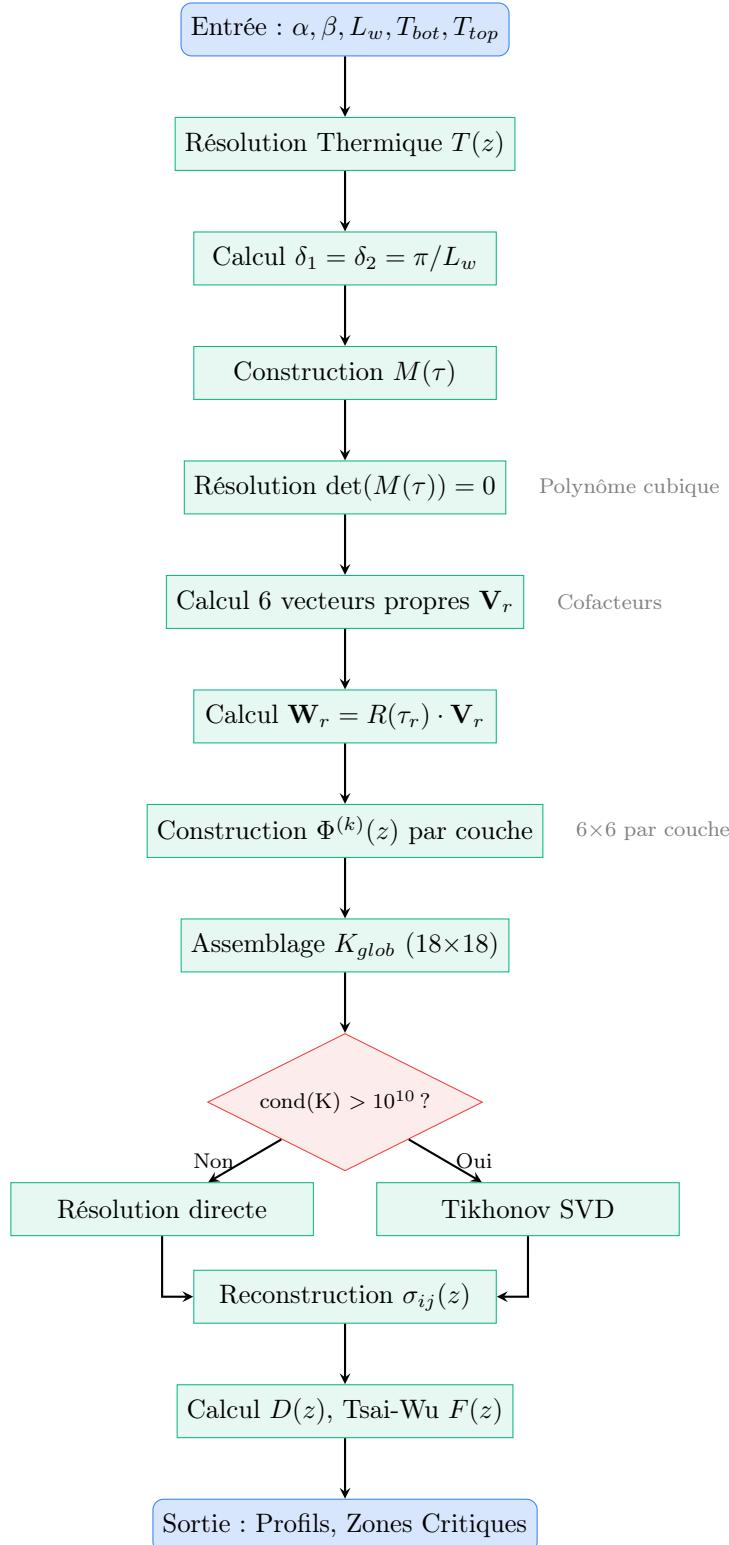
$$\alpha = \frac{h_3}{h_1} \quad (\text{ratio épaisseur céramique/substrat, typiquement 0.1 à 1.0}) \quad (1)$$

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

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

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

1.4 Organigramme de l'Algorithmme Spectral



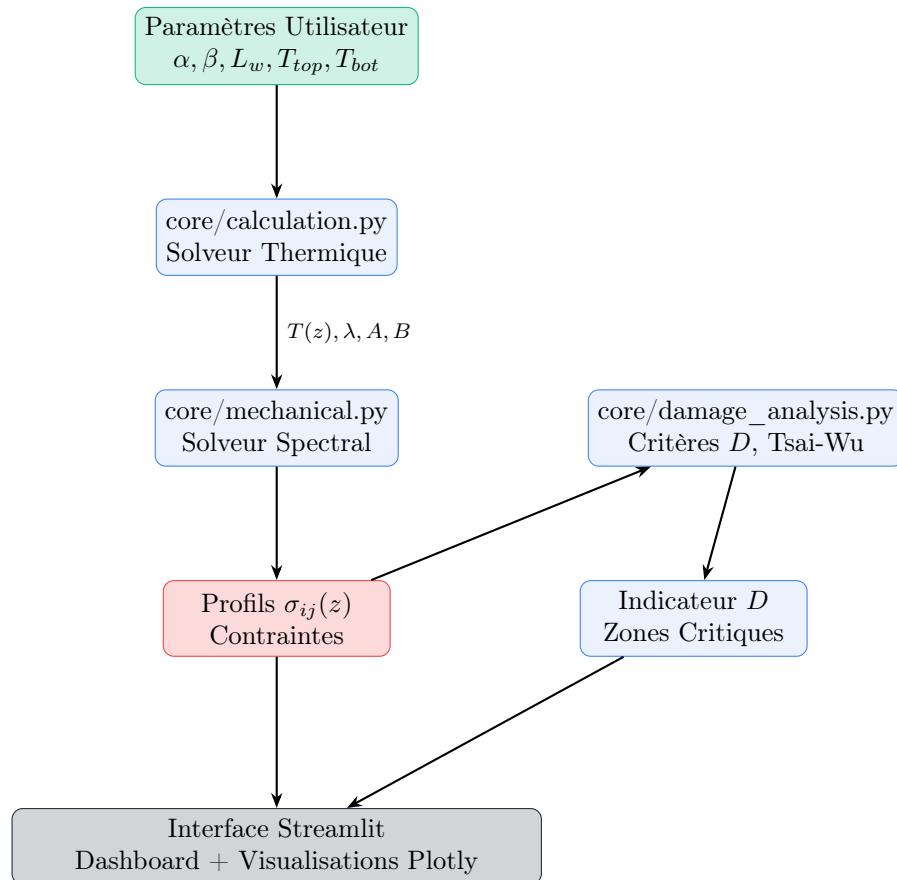
2 Architecture Globale du Code

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

```

1 import streamlit as st
2
3 st.set_page_config(
4     page_title="TBC Analysis Dashboard",
5     page_icon="shield",
6     layout="wide",
7     initial_sidebar_state="expanded"
8 )
9
10 # Import des onglets
11 from tabs import dashboard_home, mechanical, optimization, study_parametric
12
13 # Navigation par onglets
14 tab_home, tab_mech, tab_optim, tab_study = st.tabs([
15     "Dashboard", "Mecanique", "Optimisation", "Etude Parametrique"
16 ])
17
18 with tab_home:
19     dashboard_home.render()
20 with tab_mech:
21     mechanical.render()
22 # ...

```

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

```

1 @st.cache_data
2 def compute_real_damage_indicator(alpha, lw, t_top, t_bottom):
3     """
4         Calcule l'indicateur d'endommagement D base sur la physique reelle.
5
6         Logique:
7         1. Si T_interface > T_critique -> D > 1 (dommage thermique garanti)
8         2. Sinon: D = D_mecanique + bonus si proche de T_critique
9     """
10    from core.damage_analysis import CRITICAL_STRESS
11    from core.constants import GPa_TO_PA
12
13    # Modele de resistances thermiques en serie
14    R1 = h1 / CONSTANTS['k33_1']
15    R2 = h2 / CONSTANTS['k33_2']
16    R3 = h3 / CONSTANTS['k33_3']
17    R_total = R1 + R2 + R3
18
19    # Temperature a l'interface substrat/bondcoat
20    T_h1 = t_bottom + delta_T_total * R1 / R_total
21
22    # CAS 1: TEMPERATURE AU-DESSUS DE LA LIMITE CRITIQUE
23    if T_h1 > T_crit:
24        D_thermal = 1.0 + (T_h1 - T_crit) / 200.0
25        return max(0.05, min(1.5, D_thermal))
26
27    # CAS 2: D base sur les contraintes mecaniques

```

```

28     sigma_thermal = E_ceramic * delta_alpha * delta_T_interface
29     D_mechanical = max(D_ceramic_tension, D_ceramic_shear, D_bondcoat)
30
31     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 :

```

1 def run_full_analysis(h_tbc, h_bc_unused, T_hat, Lw, method, show_math,
2                       alpha, beta, t_bottom, t_top, n_modes=1):
3     """Lance l'analyse complète et stocke les résultats."""
4
5     from core.constants import PROPS_SUBSTRATE, PROPS_BONDCOAT, PROPS_CERAMIC
6     from core.constants import ALPHA_SUBSTRATE, ALPHA_BONDCOAT, ALPHA_CERAMIC
7
8     # Configuration des couches avec propriétés DISTINCTES
9     layer_configs = [
10         (h_sub_m, PROPS_SUBSTRATE, ALPHA_SUBSTRATE),           # Substrat Nickel
11         (h_bc_m, PROPS_BONDCOAT, ALPHA_BONDCOAT),             # Bond Coat MCrAlY
12         (h_tbc_m, PROPS_CERAMIC, ALPHA_CERAMIC)                # Ceramique YSZ
13     ]
14
15     # === ETAPE 7: Resolution equation caractéristique ===
16     char_eq_result = solve_characteristic_equation(delta1, delta2, props)
17     tau_roots = char_eq_result['tau_roots']
18
19     # === ETAPE 8.1: Vecteurs propres de déplacement ===
20     eigenvectors = compute_all_eigenvectors(tau_roots, delta1, delta2, props)
21
22     # === ETAPE 8.3: Vecteurs propres de contrainte ===
23     eigenvectors_with_stress = compute_all_stress_eigenvectors(
24         eigenvectors, delta1, delta2, props
25     )
26
27     # === Resolution multicouche complète ===
28     spectral_res = solve_multilayer_problem(
29         layer_configs, Lw, lambda_th, T_hat_list, method='spectral',
30     )
31
32     # === SUPERPOSITION CLT + Spectral ===
33     clt_res = solve_multilayer_clt(layer_configs, T_mean_struct)
34
35     # Stockage dans session_state pour affichage
36     st.session_state["mech_spectral_results"] = {
37         'tau_roots': tau_roots,
38         'eigenvectors': eigenvectors_with_stress,
39         'full_results': {'stress_profile': stress_total},
40         '# ...'
41     }

```

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 :

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

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 :

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

avec l'exposant thermique :

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

4.3 Étape 4 : Loi de Comportement Thermo-Élastique

Loi de Hooke avec Effet Thermique

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

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

Correspondance Notation Tensorielle \leftrightarrow Voigt :

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

4.4 Étape 5 : Ansatz de Déplacement

Forme des Champs de Déplacement

$$u_1(x_1, x_2, x_3) = V_1(x_3) \cos(\delta_1 x_1) \sin(\delta_2 x_2) \quad (9)$$

$$u_2(x_1, x_2, x_3) = V_2(x_3) \sin(\delta_1 x_1) \cos(\delta_2 x_2) \quad (10)$$

$$u_3(x_1, x_2, x_3) = V_3(x_3) \sin(\delta_1 x_1) \sin(\delta_2 x_2) \quad (11)$$

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 $M(\tau)$ et Valeurs Propres

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

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

Matrice $M(\tau)$ Complète

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

Définition des termes K_{ij} :

$$K_{11} = C_{11}\delta_1^2 + C_{66}\delta_2^2 \quad (14)$$

$$K_{12} = (C_{12} + C_{66})\delta_1\delta_2 \quad (15)$$

$$K_{13} = (C_{13} + C_{55})\delta_1 \quad (16)$$

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 :

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

4.6 Étape 7 : Matrice $R(\tau)$ 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 :

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

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 $\partial_{x_1} \cos(\delta_1 x_1) = -\delta_1 \sin(\dots)$ 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**

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

Vecteur d'État :

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

Système Global pour N couches :

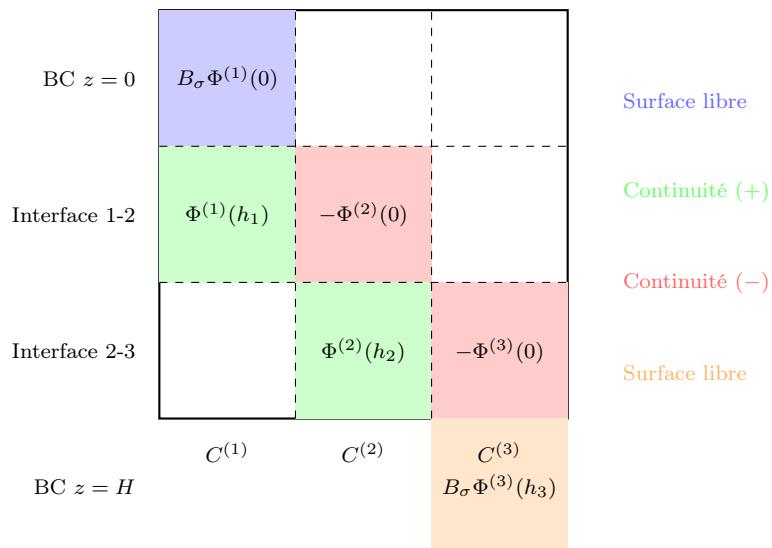
$$K_{glob} \cdot \mathbf{C}_{global} = \mathbf{F}_{thermique} \quad (21)$$

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

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

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 :

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

5 Correspondance Théorie \leftrightarrow Code : Implémentation Python

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

Référence PDF : Étapes 1-3

Résolution de la conduction thermique par couche :

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

```

1 def solve_tbc_model_v2(alpha, beta, lw, t_bottom, t_top, n_modes=1):
2     """
3         Resout le modèle thermique multicouche.
4
5     Returns:
6         dict avec T_at_h1, h3, profile_params (coeffs A, B, lambdas)
7     """
8     # Calcul des exposants thermiques par couche
9     delta_eta = np.pi / lw
10    lambda_1 = delta_eta * np.sqrt(k11_1 / k33_1)
11    lambda_2 = delta_eta * np.sqrt(k11_2 / k33_2)
12    lambda_3 = delta_eta * np.sqrt(k11_3 / k33_3)
13
14    # Resolution du système linéaire pour A_i, B_i
15    # Conditions: continuité T, continuité flux k33*dT/dx3
16    # ...
17
18    return {
19        'success': True,
20        'T_at_h1': T_interface,
21        'h3': h3,
22        'profile_params': {
23            'coeffs': [A1, B1, A2, B2, A3, B3],
24            'lambdas': [lambda_1, lambda_2, lambda_3],
25            'interfaces': [x_i1, x_i2]
26        }
27    }

```

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

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

```

1 # Substrat : Superalliage Nickel (Inconel 718)
2 PROPS_SUBSTRATE = {
3     'C11': 259.6,  # GPa - Ref ONERA Table 3
4     'C12': 179.0,
5     'C13': 179.0,
6     'C22': 259.6,
7     'C23': 179.0,
8     'C33': 259.6,
9     'C44': 109.6,  # Cisaillement
10    'C55': 109.6,
11    'C66': 40.3,
12}
13 ALPHA_SUBSTRATE = {'alpha_1': 13e-6, 'alpha_2': 13e-6, 'alpha_3': 13e-6}  # K^-1
14
15 # Ceramique : Zirconie Stabilisée Yttrium (YSZ)
16 PROPS_CERAMIC = {
17     'C11': 50.0,  'C12': 10.0,  'C13': 10.0,
18     'C22': 50.0,  'C23': 10.0,  'C33': 50.0,
19     'C44': 20.0,  'C55': 20.0,  'C66': 20.0,
20}
21 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 : $M(\tau) \cdot \mathbf{A} = 0$

```

1 def get_M_matrix(tau, delta1, delta2, props=MECHANICAL_PROPS):
2     """
3         Construit la matrice dynamique M(tau) 3x3.
4
5         THEORIE (PDF Etape 6):
6         M(tau) encapsule l'équation d'équilibre div(sigma) = 0
7         avec l'ansatz u_i = V_i * exp(tau * x3) * sin(delta * x)
8         """
9         # Termes K (independants de tau)
10        K11 = props['C11'] * delta1**2 + props['C66'] * delta2**2
11        K12 = (props['C12'] + props['C66']) * delta1 * delta2
12        K13_coeff = (props['C13'] + props['C55']) * delta1
13        K22 = props['C66'] * delta1**2 + props['C22'] * delta2**2
14        K23_coeff = (props['C23'] + props['C44']) * delta2
15        K33 = props['C55'] * delta1**2 + props['C44'] * delta2**2
16
17        M = np.zeros((3, 3), dtype=complex)
18
19        # Ligne 1: Equilibre selon x1
20        M[0, 0] = props['C55'] * tau**2 - K11
21        M[0, 1] = -K12
22        M[0, 2] = K13_coeff * tau
23
24        # Ligne 2: Equilibre selon x2
25        M[1, 0] = -K12
26        M[1, 1] = props['C44'] * tau**2 - K22
27        M[1, 2] = K23_coeff * tau
28
29        # Ligne 3: Equilibre selon x3 (SIGNES NEGATIFS - Eq. 166 PDF)
30        M[2, 0] = -K13_coeff * tau      # -(C13+C55)*delta1*tau
31        M[2, 1] = -K23_coeff * tau      # -(C23+C44)*delta2*tau
32        M[2, 2] = props['C33'] * tau**2 - K33
33
34    return M

```

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

```

1 def solve_characteristic_equation(delta1, delta2, props=MECHANICAL_PROPS):
2     """
3         Trouve les racines tau du determinant det(M(tau)) = 0.
4
5         METHODE NUMERIQUE ROBUSTE:
6         Au lieu des formules analytiques complexes, on évalue le déterminant
7         en 3 points pour identifier les coefficients du polynôme.
8         """
9
10        # Coefficient c6 (analytique exact - Eq. 18 PDF)
11        c6 = props['C55'] * props['C44'] * props['C33']
12
13        def get_det_at_X(X_val):
14            """Évalue det(M(sqrt(X))) pour construire le polynôme."""
15            tau_val = np.sqrt(complex(X_val))
16            M = get_M_matrix(tau_val, delta1, delta2, props)
17            return compute_determinant_gaussian(M)
18
19        # Evaluations en X = tau^2
20        P_0 = get_det_at_X(0)      # P(0) = c0
21        P_1 = get_det_at_X(1)      # P(1) = c6 + c4 + c2 + c0
22        P_2 = get_det_at_X(2)      # P(2) = 8c6 + 4c4 + 2c2 + c0
23
24        # Système linéaire pour c4, c2
25        c0 = P_0
26        b1 = P_1 - c6 - c0
27        b2 = P_2 - 8*c6 - c0
28        c4 = (b2 - 2*b1) / 2
29        c2 = b1 - c4
30
31        # Racines du polynôme cubique en X

```

```

31     coeffs_norm = [1, c4/c6, c2/c6, c0/c6]
32     X_roots = np.roots(coeffs_norm)
33
34     # 6 racines tau = +/- sqrt(X)
35     tau_roots = []
36     for X in X_roots:
37         tau_roots.extend([np.sqrt(X), -np.sqrt(X)])
38
39     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

```

1 def get_R_matrix(tau, delta1, delta2, props=MECHANICAL_PROPS):
2     """
3         Construit la matrice R(tau) 3x3 pour le calcul du vecteur contrainte.
4
5     DERIVATION (conforme PDF Etape 7):
6     sigma_13 = C55 * (tau*V1 + delta1*V3)
7     sigma_23 = C44 * (tau*V2 + delta2*V3)
8     sigma_33 = -C13*delta1*V1 - C23*delta2*V2 + C33*tau*V3
9
10    Les signes NEGATIFS proviennent de d/dx1(cos) = -delta1*sin
11    """
12    R = np.zeros((3, 3), dtype=complex)
13
14    # sigma_13 = C55(tau V1 + delta1 V3)
15    R[0, 0] = props['C55'] * tau
16    R[0, 1] = 0
17    R[0, 2] = props['C55'] * delta1
18
19    # sigma_23 = C44(tau V2 + delta2 V3)
20    R[1, 0] = 0
21    R[1, 1] = props['C44'] * tau
22    R[1, 2] = props['C44'] * delta2
23
24    # sigma_33 = -C13*delta1*V1 - C23*delta2*V2 + C33*tau*V3
25    R[2, 0] = -props['C13'] * delta1 # Signe NEGATIF CONFIRME
26    R[2, 1] = -props['C23'] * delta2 # Signe NEGATIF CONFIRME
27    R[2, 2] = props['C33'] * tau
28
29    return R

```

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

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

```

1 def solve_multilayer(layers, delta1, delta2, lambda_th=None, T_hat=None):
2     """
3         Resout le probleme mecanique pour un systeme multicouche.
4
5         IMPLEMENTATION AVEC PRECONDITIONNEMENT:
6         1. Construction du systeme K_glob @ C = F
7         2. Equilibrage par scaling: D_r @ K_glob @ D_c @ y = D_r @ F
8         3. Resolution du systeme equilibre
9         4. De-scaling: C = D_c @ y
10
11    N = len(layers)
12
13    # Setup de chaque couche (modes propres + forage thermique)
14    for k, layer in enumerate(layers):
15        setup_layer(layer, delta1, delta2, th_data)
16
17    # Matrice globale 6N x 6N
18    K_glob = np.zeros((6*N, 6*N), dtype=complex)
19    F_glob = np.zeros(6*N, dtype=complex)
20
21    # Matrice de selection des contraintes
22    B_stress = np.array([
23        [0, 0, 0, 1, 0, 0],

```

```

24     [0, 0, 0, 0, 1, 0],
25     [0, 0, 0, 0, 0, 1]
26 ], dtype=complex)
27
28 # BLOC 1: BC en z=0 (surface libre)
29 Phi_0 = build_Phi_matrix_normalized(0, layers[0].eigenvectors)
30 K_glob[0:3, 0:6] = B_stress @ Phi_0
31
32 # BLOCS de continuité aux interfaces
33 for k in range(N - 1):
34     Phi_k_top = build_Phi_matrix_normalized(layers[k].h, layers[k].eigenvectors)
35     Phi_kp1_bot = build_Phi_matrix_normalized(0, layers[k+1].eigenvectors)
36     K_glob[row_idx:row_idx+6, 6*k:6*(k+1)] = Phi_k_top
37     K_glob[row_idx:row_idx+6, 6*(k+1):6*(k+2)] = -Phi_kp1_bot
38
39 # BLOC N: BC en z=H (surface libre)
40 Phi_H = build_Phi_matrix_normalized(layers[-1].h, layers[-1].eigenvectors)
41 K_glob[-3:, -6:] = B_stress @ Phi_H
42
43 # PRECONDITIONNEMENT PAR EQUILIBRAGE
44 row_scales = [1.0 / np.max(np.abs(K_glob[i, :])) for i in range(6*N)]
45 D_r = np.diag(row_scales)
46 K_scaled = D_r @ K_glob
47 F_scaled = D_r @ F_glob
48
49 # Resolution
50 y, solve_info = solve_regularized_system(K_scaled @ D_c, F_scaled)
51 C_total = D_c @ y # Descaling
52
53 return {'layers': layers, 'C_total': C_total, 'cond_K': solve_info['cond']}

```

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

5.6 Nouvelle Implémentation : Module mechanical_pdf.py

Un nouveau module core/mechanical_pdf.py a été développé pour suivre **exactement** la méthodologie du document `equilibre_local_corrige.pdf`, section par section.

Structure du Module mechanical_pdf.py

Le module est organisé selon les sections du PDF :

- **Section 2** : Forme de la solution générale $U_\alpha(x_3)$
- **Section 5** : Opérateurs L_{jk} et termes thermiques Q_α
- **Section 6** : Matrice dynamique $\Gamma(\tau)$
- **Section 7** : Assemblage matriciel 9×9 bloc-diagonal

5.6.1 Section 2 : Forme de la Solution Générale

Champ de Déplacement

$$U_\alpha(x_3) = \sum_{r=1}^3 A_\alpha^r \cdot e^{\tau_r \cdot x_3}, \quad \alpha \in \{1, 2, 3\} \quad (24)$$

où les τ_r sont les 3 valeurs propres sélectionnées ($\text{Re}(\tau) < 0$).

```

1 def build_displacement_field(A_amplitudes, tau_roots, x3):
2     """
3     Construit le champ de déplacement U(x3).
4
5     REFERENCE: equilibre_local_corrige.pdf, Section 2
6     """
7     U = np.zeros(3, dtype=complex)
8     for alpha in range(3):
9         for r in range(3):
10             U[alpha] += A_amplitudes[alpha, r] * np.exp(tau_roots[r] * x3)
11

```

Listing 10 – Solution Générale (core/mechanical_pdf.py)

5.6.2 Section 5 : Opérateurs L_{jk} et Termes ThermiquesOpérateurs L_{jk} de la Matrice Dynamique

Termes diagonaux :

$$L_{11} = C_{55}\tau^2 - (C_{11}\delta_1^2 + C_{66}\delta_2^2) \quad (25)$$

$$L_{22} = C_{44}\tau^2 - (C_{22}\delta_2^2 + C_{66}\delta_1^2) \quad (26)$$

$$L_{33} = C_{33}\tau^2 - (C_{55}\delta_1^2 + C_{44}\delta_2^2) \quad (27)$$

Termes croisés (symétriques) :

$$L_{12} = L_{21} = -(C_{12} + C_{66})\delta_1\delta_2 \quad (28)$$

Termes hors-plan (antisymétriques) :

$$L_{13} = +(C_{13} + C_{55})\delta_1\tau, \quad L_{31} = -L_{13} \quad (29)$$

$$L_{23} = +(C_{23} + C_{44})\delta_2\tau, \quad L_{32} = -L_{23} \quad (30)$$

```

1 def compute_L_operators(tau, delta1, delta2, props):
2     """
3         Calcule les opérateurs L_jk de la matrice dynamique.
4
5         REFERENCE: equilibre_local_corrige.pdf, Section 5
6         """
7
8     # Termes diagonaux
9     L_11 = props['C55'] * tau**2 - (props['C11'] * delta1**2 + props['C66'] * delta2**2)
10    L_22 = props['C44'] * tau**2 - (props['C22'] * delta2**2 + props['C66'] * delta1**2)
11    L_33 = props['C33'] * tau**2 - (props['C55'] * delta1**2 + props['C44'] * delta2**2)
12
13    # Termes croises (symetriques)
14    L_12 = L_21 = -(props['C12'] + props['C66']) * delta1 * delta2
15
16    # Termes hors-plan (ANTISYMETRIQUES!)
17    L_13 = +(props['C13'] + props['C55']) * delta1 * tau
18    L_31 = -L_13 # Signe negatif!
19    L_23 = +(props['C23'] + props['C44']) * delta2 * tau
20    L_32 = -L_23 # Signe negatif!
21
22    return {'L_11': L_11, 'L_12': L_12, ..., 'L_33': L_33}

```

Listing 11 – Opérateurs L_{jk} (core/mechanical_pdf.py)Termes Thermiques Q_α

$$Q_1 = (C_{11}\alpha_{11} + C_{12}\alpha_{22})\delta_1 \cdot T(x_3 = \bar{h}) \quad (31)$$

$$Q_2 = (C_{22}\alpha_{22} + C_{12}\alpha_{11})\delta_2 \cdot T(x_3 = \bar{h}) \quad (32)$$

$$Q_3 = (C_{13}\alpha_{11} + C_{23}\alpha_{22} + C_{33}\alpha_{33}) \cdot \left. \frac{dT}{dx_3} \right|_{x_3=\bar{h}} \quad (33)$$

```

1 def compute_Q_thermal_vector(delta1, delta2, T, dT_dx3, alpha_coeffs, props):
2     """
3         Calcule le vecteur de sollicitation thermique Q = [Q1, Q2, Q3].
4
5         REFERENCE: equilibre_local_corrige.pdf, Section 7 (Eq. 266-269)
6         """

```

```

7   Q1 = (props['C11']*alpha['alpha_1'] + props['C12']*alpha['alpha_2']) * delta1 * T
8   Q2 = (props['C22']*alpha['alpha_2'] + props['C12']*alpha['alpha_1']) * delta2 * T
9   Q3 = (props['C13']*alpha['alpha_1'] + props['C23']*alpha['alpha_2'])
10  + props['C33']*alpha['alpha_3']) * dT_dx3
11  return np.array([Q1, Q2, Q3])

```

Listing 12 – Termes Thermiques Q (core/mechanical_pdf.py)

5.6.3 Section 6 : Matrice $\Gamma(\tau)$

Matrice Dynamique $\Gamma(\tau)$

Le système homogène s'écrit $\Gamma(\tau) \cdot \mathbf{A} = \mathbf{0}$ avec :

$$\Gamma(\tau) = \begin{pmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \quad (34)$$

Propriétés :

- $\Gamma_{12} = \Gamma_{21}$ (symétrique)
- $\Gamma_{13} = -\Gamma_{31}$ et $\Gamma_{23} = -\Gamma_{32}$ (antisymétrique)

```

1 def get_Gamma_matrix(tau, delta1, delta2, props):
2     """
3         Construit la matrice dynamique Gamma(tau) 3x3.
4
5         REFERENCE: equilibre_local_corrige.pdf, Section 6 (Eq. 199-208)
6     """
7     L = compute_L_operators(tau, delta1, delta2, props)
8     return np.array([
9         [L['L_11'], L['L_12'], L['L_13']],
10        [L['L_21'], L['L_22'], L['L_23']],
11        [L['L_31'], L['L_32'], L['L_33']]
12    ], dtype=complex)

```

Listing 13 – Matrice Gamma (core/mechanical_pdf.py)

5.6.4 Section 7 : Assemblage Matriciel 9×9

Matrice Bloc-Diagonale K_{Dyn}

La matrice globale 9×9 est bloc-diagonale :

$$\mathbb{K}_{Dyn} = \begin{pmatrix} \Gamma(\tau_1) & 0 & 0 \\ 0 & \Gamma(\tau_2) & 0 \\ 0 & 0 & \Gamma(\tau_3) \end{pmatrix}_{9 \times 9} \quad (35)$$

Avec le vecteur d'amplitudes $\mathcal{A} = (A_1^1, A_2^1, A_3^1, A_1^2, A_2^2, A_3^2, A_1^3, A_2^3, A_3^3)^T$.

```

1 def assemble_K_dyn_9x9(tau_roots, delta1, delta2, props):
2     """
3         Assemble la matrice dynamique bloc-diagonale 9x9.
4
5         REFERENCE: equilibre_local_corrige.pdf, Section 7 (Eq. 220-235)
6     """
7     K_dyn = np.zeros((9, 9), dtype=complex)
8
9     for r in range(3):
10        Gamma_r = get_Gamma_matrix(tau_roots[r], delta1, delta2, props)
11        start, end = 3*r, 3*(r+1)
12        K_dyn[start:end, start:end] = Gamma_r

```

```

13     return K_dyn
14

```

Listing 14 – Assemblage 9x9 (core/mechanical_pdf.py)

Système Final :

$$\mathbb{K}_{Dyn} \cdot \mathcal{A} = \mathcal{F}_{Th} \quad (36)$$

où $\mathcal{F}_{Th} = (Q_1, Q_2, Q_3, Q_1, Q_2, Q_3, Q_1, Q_2, Q_3)^T$.

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

```

1 def compute_damage_indicator(sigma_13, sigma_23, sigma_33, layer_type='ceramic',
2                             sigma_11=None, sigma_22=None):
3     """
4         Calcule l'indicateur d'endommagement D pour chaque point.
5
6     D = max(|sigma_ij| / sigma_crit_ij)
7
8     D < 1: Sur
9     D = 1: Limite
10    D > 1: Endommagement probable
11    """
12    crit = CRITICAL_STRESS.get(layer_type, CRITICAL_STRESS['ceramic'])
13
14    # D cisaillement transverse
15    D_shear = np.maximum(np.abs(sigma_13), np.abs(sigma_23)) / crit['sigma_shear']
16
17    # D normal transverse (sigma_33)
18    D_33 = np.where(
19        sigma_33 >= 0,
20        np.abs(sigma_33) / crit['sigma_tensile'],
21        np.abs(sigma_33) / crit['sigma_compressive']
22    )
23
24    D = np.maximum(D_shear, D_33)
25
26    # CRITIQUE: Inclure les contraintes planes sigma_11, sigma_22
27    if sigma_11 is not None:
28        D_11 = np.abs(sigma_11) / crit['sigma_tensile']
29        D = np.maximum(D, D_11)
30
31    return D

```

Listing 15 – Indicateur de Dommage D (lignes 38-89)

```

1 def compute_tsai_wu_criterion(sigma_13, sigma_23, sigma_33, layer_type='ceramic'):
2     """
3         Critère de Tsai-Wu pour matériaux composites/anisotropes.
4
5         F = F_i * sigma_i + F_ij * sigma_i * sigma_j
6
7         F >= 1: Rupture
8     """
9
10    crit = CRITICAL_STRESS.get(layer_type)
11
12    # Coefficients F_i (asymétrie traction/compression)
13    F_3 = 1/crit['sigma_tensile'] - 1/crit['sigma_compressive']
14
15    # Coefficients F_ij
16    F_33 = 1 / (crit['sigma_tensile'] * crit['sigma_compressive'])
17    F_44 = 1 / (crit['sigma_shear']**2)
18    F_55 = 1 / (crit['sigma_shear']**2)
19
20    F = F_3 * sigma_33 + F_33 * sigma_33**2 + F_44 * sigma_23**2 + F_55 * sigma_13**2
21
22    return F

```

Listing 16 – 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 :20	Calcul \mathbf{V}_r par cofacteurs
Étape 7 : Matrice R	get_R_matrix()	mechanical.py :26	Passage $\mathbf{V} \rightarrow \mathbf{W}$
Étape 7 : Vecteurs W	compute_stress_eigenvector()	mechanical.py :33	$\mathbf{W}_r = R \cdot \mathbf{V}_r$
Étape 8 : Matrice $\Phi(z)$	build_Phi_matrix()	mechanical.py :47	Assemblage 6×6
Étape 8 : Système global	solve_multilayer()	mechanical.py :99	$\mathbf{K}_{glob} \cdot \mathbf{C} = \mathbf{F}$
Étape 8 : Préconditionnement	solve_regularized_system()	mechanical.py :84	Tikhonov + scaling
Reconstruction	compute_multilayer_stress_profile()	mechanical.py :11	Profils $\sigma_{ij}(z)$
Endommagement D	compute_damage_indicator()	mechanical.py :38	$D = \max(\sigma /\sigma_{crit})$
Critère Tsai-Wu	compute_tsai_wu_criterion()	mechanical.py :40	Si $F \geq 1$
Interface Streamlit			
Dashboard	render()	dashboard_home.py	KPIs, 3D
Analyse Mécanique	run_full_analysis()	tabs/mechanical.py	Workflow complet
Affichage Résultats	display_spectral_results()	tabs/mechanical.py	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

$$\beta_i = \sum_{j=1}^3 C_{ij} \alpha_j = C_{i1}\alpha_1 + C_{i2}\alpha_2 + C_{i3}\alpha_3 \quad (37)$$

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

$$\beta_1 = (C_{11} + C_{12} + C_{13})\alpha \quad (38)$$

$$\beta_2 = (C_{21} + C_{22} + C_{23})\alpha \quad (39)$$

$$\beta_3 = (C_{31} + C_{32} + C_{33})\alpha \quad (40)$$

```

1 def compute_beta_coefficients(alpha_coeffs, props=MECHANICAL_PROPS):
2     """
3         Calcule les coefficients de contrainte thermique beta.
4         beta_i = C_i1*alpha_1 + C_i2*alpha_2 + C_i3*alpha_3
5     """
6     if isinstance(alpha, dict):
7         a1 = alpha.get('alpha_1', 10e-6)
8         a2 = alpha.get('alpha_2', a1)
9         a3 = alpha.get('alpha_3', a1)

```

```

10     else:
11         a1 = a2 = a3 = alpha
12
13     beta_1 = props['C11']*a1 + props['C12']*a2 + props['C13']*a3
14     beta_2 = props['C12']*a1 + props['C22']*a2 + props['C23']*a3
15     beta_3 = props['C13']*a1 + props['C23']*a2 + props['C33']*a3
16
17     return np.array([beta_1, beta_2, beta_3])

```

Listing 17 – 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

$$\mathbf{F}_{th} = \hat{T} \begin{pmatrix} \beta_1 \delta_1 \\ \beta_2 \delta_2 \\ \beta_3 \lambda_{th} \end{pmatrix} \quad (41)$$

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

$$D_r \cdot K_{glob} \cdot D_c \cdot \mathbf{y} = D_r \cdot \mathbf{F} \quad (42)$$

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

$$D_r[i, i] = \frac{1}{\max_j |K[i, j]|} \quad (43)$$

$$D_c[j, j] = \frac{1}{\max_i |K_{scaled}[i, j]|} \quad (44)$$

9.2 Régularisation de Tikhonov

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

$$K = U \Sigma V^H \quad (45)$$

La solution régularisée est :

$$\mathbf{x}_{reg} = \sum_{i=1}^n \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} \cdot \frac{\mathbf{u}_i^H \mathbf{b}}{\sigma_i} \cdot \mathbf{v}_i \quad (46)$$

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

```

1 def solve_regularized_system(K, F, tol=1e-12):
2     """Resolution robuste avec regularisation de Tikhonov adaptative."""
3     cond_K = np.linalg.cond(K)
4
5     if cond_K < 1e10:
6         # Systeme bien conditionne - resolution directe
7         x = np.linalg.solve(K, F)
8     else:
9         # Decomposition SVD
10        U, s, Vh = np.linalg.svd(K)
11
12        # Estimation du parametre lambda par GCV simplifie
13        lambda_reg = s[k_noise] * np.sqrt(noise_to_signal)
14
15        # Facteurs de filtrage
16        filter_factors = s**2 / (s**2 + lambda_reg**2)
17
18        # Solution regularisee
19        x = Vh.conj().T @ (filter_factors * (U.conj().T @ F) / s)
20
21    return x, {'cond': cond_K, 'regularized': cond_K >= 1e10}

```

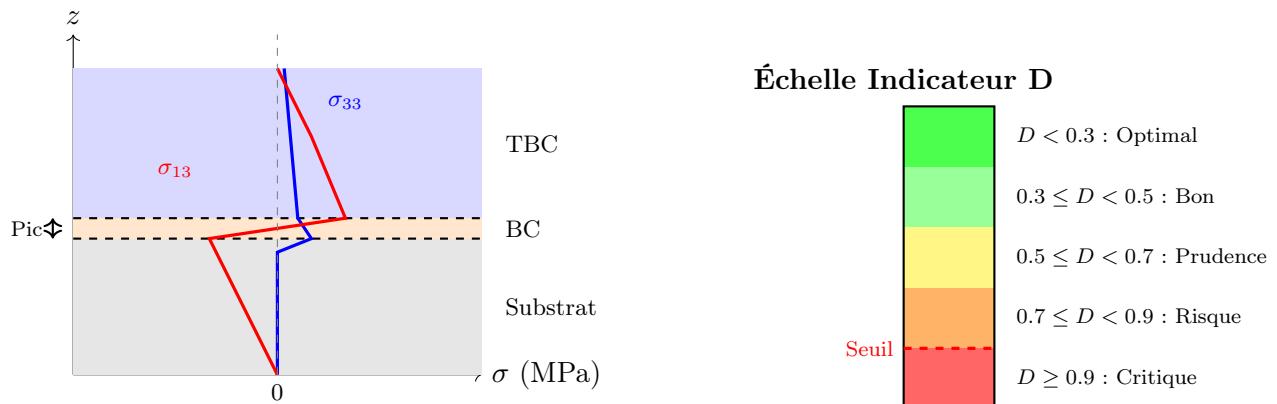
Listing 18 – Régularisation Tikhonov (core/mechanical.py lignes 846-989)

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



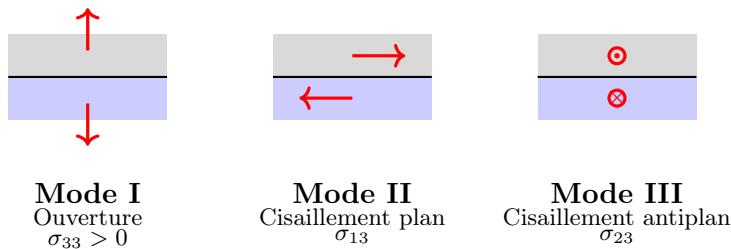
10.2 Zones Critiques Typiques

- **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

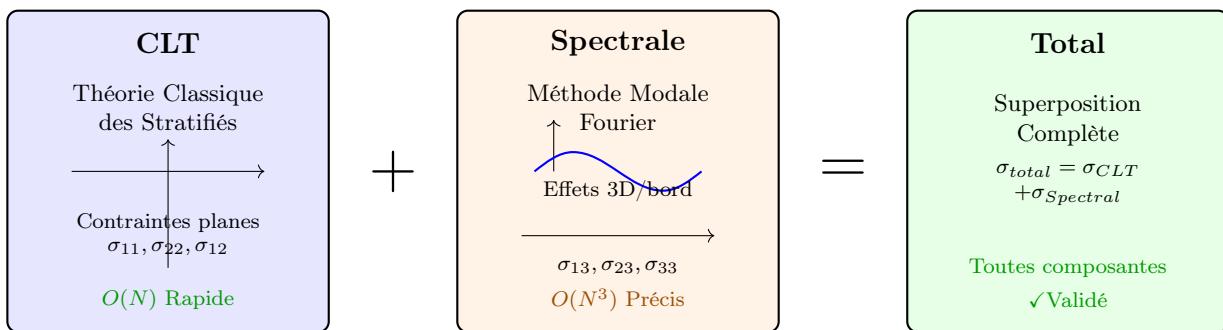
Action	Effet sur σ_{max}	Tendance D
$\alpha \uparrow$ (TBC plus épais)	Gradient thermique plus étalé, meilleure isolation	\downarrow
$L_w \downarrow$ (perturbation courte)	Gradients latéraux plus forts	\uparrow
$\Delta T \uparrow$	Forçage thermique linéairement croissant	\uparrow
$\beta \downarrow$ (TBC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

10.4 Modes de Rupture aux Interfaces



11 Comparaison : Méthode Spectrale vs CLT

Le code implémente deux méthodes complémentaires :



Méthode Spectrale (Principale)	CLT (Classical Theory)	Laminate
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 :

$$\sigma_{total} = \sigma_{CLT} + \sigma_{Spectral} \quad (47)$$

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

12 Validation : Référence ONERA/Safran

Les propriétés matériau 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 **träçabilité 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 $T_{interface} < T_{crit} = 1100^\circ\text{C}$
- Augmenter α (épaisseur TBC) pour réduire les contraintes aux interfaces