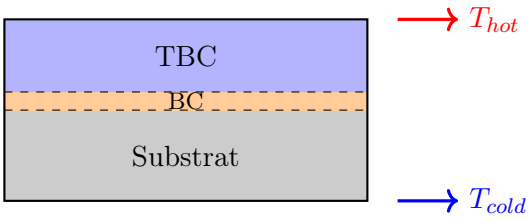
# Rapport de Synthèse Exhaustif

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

# 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) |
| $\sigma_{33}$ 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

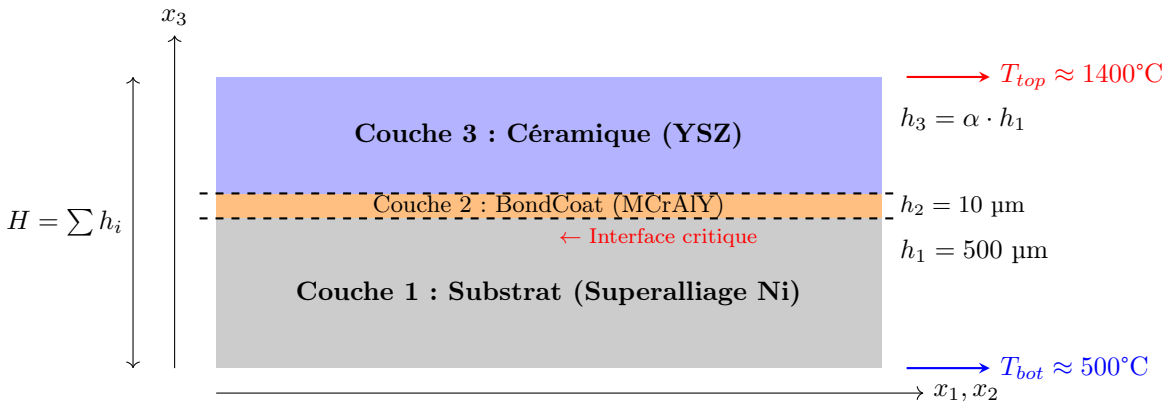
Ce document établit la correspondance **rigoureuse** entre :
— Le document théorique de référence `ProjectEstaca.pdf` (8 étapes)
— L'implémentation Python dans le module `core/`
— L'interface utilisateur Streamlit dans le module `tabs/`

## 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 :

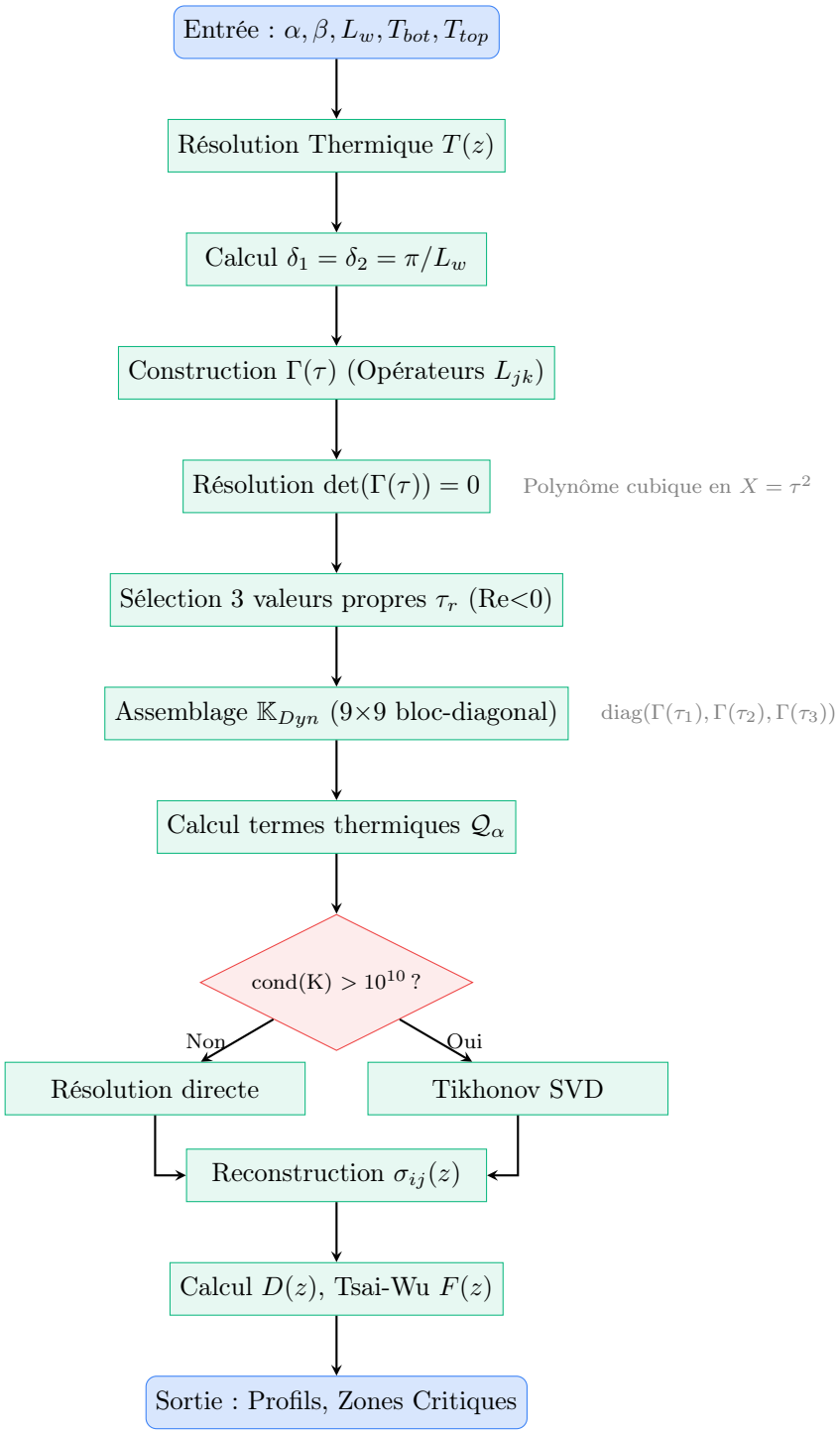
$$\alpha = \frac{h_3}{h_1} \quad \text{(ratio épaisseur céramique/substrat, typiquement 0.1 à 1.0)} \tag{1}$$

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

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

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

### 1.4 Organigramme de l'Algorithme Spectral




## 2 Architecture Globale du Code

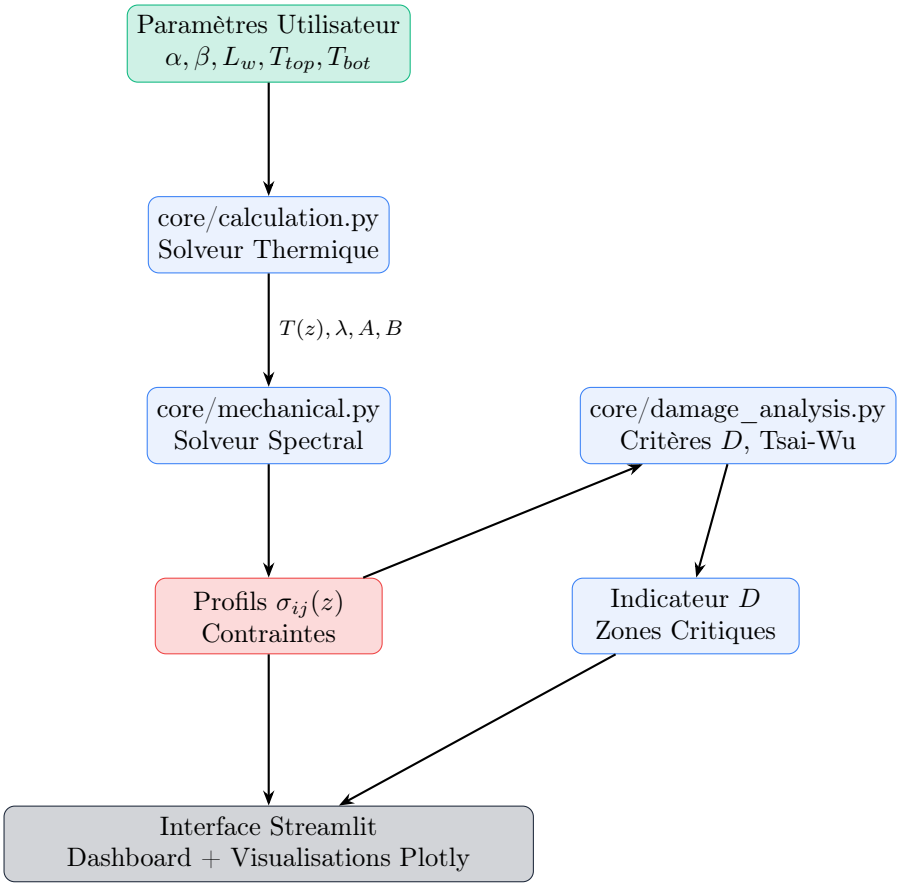
### 2.1 Structure des Répertoires

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 $\Delta T$, 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

```
@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."""

```

```python
    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 $\alpha_{kl}$ 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 $\tau$ est la **valeur propre** à déterminer.

### 4.5 Étape 6 : Matrice Dynamique $\Gamma(\tau)$ et Valeurs Propres

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

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

**Matrice $\Gamma(\tau)$ - Opérateurs $L_{jk}$**

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

**Termes diagonaux :**

$$L_{11} = C_{55}\tau^2 - (C_{11}\delta_1^2 + C_{66}\delta_2^2) \tag{14}$$
$$L_{22} = C_{44}\tau^2 - (C_{22}\delta_2^2 + C_{66}\delta_1^2) \tag{15}$$
$$L_{33} = C_{33}\tau^2 - (C_{55}\delta_1^2 + C_{44}\delta_2^2) \tag{16}$$

**Termes croisés dans le plan (symétriques) :**

$$L_{12} = L_{21} = -(C_{12} + C_{66})\delta_1\delta_2 \tag{17}$$

**Termes hors-plan (ANTISYMÉTRIQUES) :**

$$L_{13} = +(C_{13} + C_{55})\delta_1\tau, \quad L_{31} = -(C_{13} + C_{55})\delta_1\tau \tag{18}$$
$$L_{23} = +(C_{23} + C_{44})\delta_2\tau, \quad L_{32} = -(C_{23} + C_{44})\delta_2\tau \tag{19}$$

**Propriété d'Antisymétrie Critique**

$L_{13} = -L_{31}$ et $L_{23} = -L_{32}$ : cette antisymétrie provient de l'équation d'équilibre en direction $x_3$ et est **essentielle** pour la physique correcte.

**Équation Caractéristique :**

$$\det(\Gamma(\tau)) = 0 \quad \Rightarrow \quad \text{Polynôme d'ordre 6 : } P(\tau) = c_6\tau^6 + c_4\tau^4 + c_2\tau^2 + c_0 = 0 \tag{20}$$

En posant $X = \tau^2$, on obtient un polynôme cubique avec 3 racines $X_i$. On sélectionne les 3 valeurs propres $\tau_r$ avec $\mathrm{Re}(\tau_r) < 0$ (condition de radiation).

### 4.6 Étape 7 : Assemblage $9 \times 9$ et Termes Thermiques

Le système global pour une couche $(i)$ s'écrit :

$$\left[\mathbb{K}_{Dyn}^{(i)}\right]_{(9\times9)} \cdot \{\mathcal{A}^{(i)}\}_{(9\times1)} = \{\mathcal{F}_{Th}^{(i)}\}_{(9\times1)} \tag{21}$$

**Matrice Bloc-Diagonale $\mathbb{K}_{Dyn}$**

$$\boxed{\mathbb{K}_{Dyn}^{(i)} = \begin{pmatrix} \Gamma(\tau_1) & 0 & 0 \\ 0 & \Gamma(\tau_2) & 0 \\ 0 & 0 & \Gamma(\tau_3) \end{pmatrix}_{9\times9}} \tag{22}$$

Chaque bloc $\Gamma(\tau_r)$ est la matrice $3 \times 3$ des opérateurs $L_{jk}$ évaluée à $\tau = \tau_r$.

**Vecteur d'amplitudes et vecteur thermique :**

$$\mathcal{A}^{(i)} = \begin{pmatrix} A_1^1 \\ A_2^1 \\ A_3^1 \\ A_1^2 \\ A_2^2 \\ A_3^2 \\ A_1^3 \\ A_2^3 \\ A_3^3 \end{pmatrix}^{(i)}, \quad \mathcal{F}_{Th}^{(i)} = \begin{pmatrix} \mathcal{Q}_1 \\ \mathcal{Q}_2 \\ \mathcal{Q}_3 \\ \mathcal{Q}_1 \\ \mathcal{Q}_2 \\ \mathcal{Q}_3 \\ \mathcal{Q}_1 \\ \mathcal{Q}_2 \\ \mathcal{Q}_3 \end{pmatrix}^{(i)} \tag{23}$$

**Termes de Sollicitation Thermique $\mathcal{Q}_\alpha$**

$$\mathcal{Q}_1^{(i)} = (C_{11}\alpha_{11} + C_{12}\alpha_{22})\,\delta_1\,T(x_3 = \bar{h}^{(i)}) \tag{24}$$
$$\mathcal{Q}_2^{(i)} = (C_{22}\alpha_{22} + C_{12}\alpha_{11})\,\delta_2\,T(x_3 = \bar{h}^{(i)}) \tag{25}$$
$$\mathcal{Q}_3^{(i)} = (C_{13}\alpha_{11} + C_{23}\alpha_{22} + C_{33}\alpha_{33})\,\left.\frac{dT}{dx_3}\right|_{x_3=\bar{h}^{(i)}} \tag{26}$$

### 4.7 Étape 8 : Assemblage Multicouche

**Système Global pour $N$ couches :**

$$\boxed{K_{glob} \cdot \mathbf{C}_{global} = \mathbf{F}_{thermique}} \tag{27}$$

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 ✓

**Réduction du Nombre d'Équations : 27 → 18**

**Formulation théorique complète** (pour $N = 3$ couches) :

| Type d'équation | Nombre | Formule |
|---|---|---|
| Équilibre volumique (div $\sigma = 0$) | 9 | 3 directions × 3 couches |
| Continuité aux interfaces | 12 | 2 interfaces × 6 conditions |
| Conditions aux limites ($\sigma = 0$) | 6 | 2 surfaces × 3 composantes |
| **Total théorique** | **27** | |

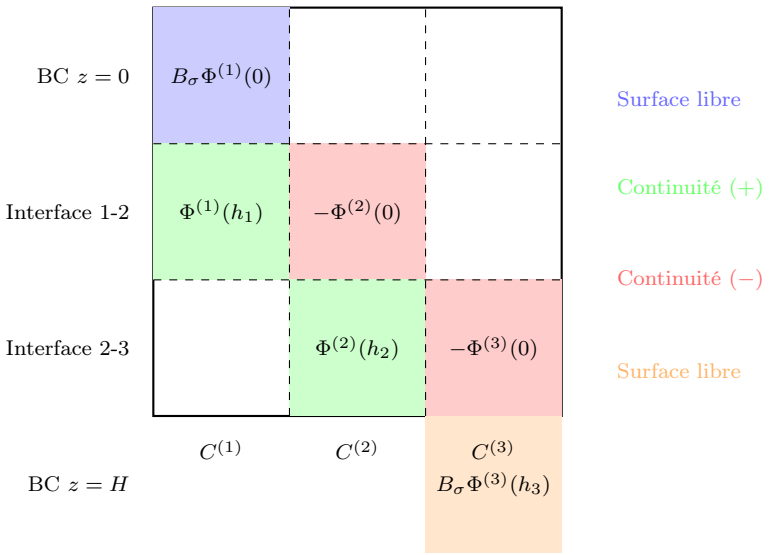
**Réduction par l'approche modale (méthode spectrale) :**
Les **9 équations d'équilibre volumique sont IMPLICITEMENT satisfaites** !
En écrivant la solution sous la forme $U_\alpha(x_3) = \sum_r A_\alpha^r \cdot e^{\tau_r x_3}$ avec les $\tau_r$ **valeurs propres** de $\Gamma(\tau)$ (définies par $\det(\Gamma(\tau_r)) = 0$), l'équation d'équilibre $\Gamma(\tau_r) \cdot \mathbf{A}^r = 0$ est automatiquement vérifiée.

**Système final résolu :**

| Type d'équation | Nombre | Implémentation |
|---|---|---|
| Équilibre volumique | 0 | ✓ Implicite (via $\tau_r$) |
| Continuité aux interfaces | 12 | $\Phi^{(k)}(h_k) = \Phi^{(k+1)}(0)$ |
| Conditions aux limites | 6 | $\sigma(0) = 0$, $\sigma(H) = 0$ |
| **Total résolu** | **18** | $= 6N$ pour $N$ couches |

**Conclusion :** La puissance de la méthode spectrale réside dans la réduction de 27 → 18 équations grâce à la structure modale.

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




### 4.8 Critères d'Endommagement

**Indicateur de Dommage D**

$$D = \max_{ij} \left( \frac{|\sigma_{ij}|}{\sigma_{crit}^{ij}} \right) \tag{28}$$

**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** | $\sigma_t$ (MPa) | $\sigma_c$ (MPa) | $\tau$ (MPa) |
|---|---|---|---|
| Substrat (Ni) | 1000 | 1200 | 600 |
| BondCoat (MCrAlY) | 500 | 700 | 300 |
| Céramique (YSZ) | 150 | 500 | 120 |

**Critère de Tsai-Wu :** Pour matériaux anisotropes :

$$F = F_3\sigma_{33} + F_{33}\sigma_{33}^2 + F_{44}\sigma_{23}^2 + F_{55}\sigma_{13}^2 \quad \text{(Rupture si } F \geq 1\text{)} \tag{29}$$

## 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 :

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

```
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)

```
# 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 : $M(\tau) \cdot \mathbf{A} = \mathbf{0}$

```
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)

```
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)

## 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 $Q_\alpha$
- **Section 6** : Matrice dynamique $\Gamma(\tau)$
- **Section 7** : Assemblage matriciel $9 \times 9$ bloc-diagonal

### 5.6.1 Section 2 : Forme de la Solution Générale

**Champ de Déplacement**

$$U_\alpha(x_3) = \sum_{r=1}^{3} A_\alpha^r \cdot e^{\tau_r \cdot x_3}, \quad \alpha \in \{1, 2, 3\} \tag{30}$$

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

```
def build_displacement_field(A_amplitudes, tau_roots, x3):
    """
    Construit le champ de deplacement U(x3).

    REFERENCE: equilibre_local_corrige.pdf, Section 2
    """
    U = np.zeros(3, dtype=complex)
    for alpha in range(3):
        for r in range(3):
            U[alpha] += A_amplitudes[alpha, r] * np.exp(tau_roots[r] * x3)
    return U
```

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

### 5.6.2 Section 5 : Opérateurs $L_{jk}$ et Termes Thermiques

**Opérateurs $L_{jk}$ de la Matrice Dynamique**

**Termes diagonaux :**

$$L_{11} = C_{55}\tau^2 - (C_{11}\delta_1^2 + C_{66}\delta_2^2) \tag{31}$$
$$L_{22} = C_{44}\tau^2 - (C_{22}\delta_2^2 + C_{66}\delta_1^2) \tag{32}$$
$$L_{33} = C_{33}\tau^2 - (C_{55}\delta_1^2 + C_{44}\delta_2^2) \tag{33}$$

**Termes croisés (symétriques) :**

$$L_{12} = L_{21} = -(C_{12} + C_{66})\delta_1\delta_2 \tag{34}$$

**Termes hors-plan (antisymétriques) :**

$$L_{13} = +(C_{13} + C_{55})\delta_1\tau, \quad L_{31} = -L_{13} \tag{35}$$
$$L_{23} = +(C_{23} + C_{44})\delta_2\tau, \quad L_{32} = -L_{23} \tag{36}$$

```python
def compute_L_operators(tau, delta1, delta2, props):
    """
    Calcule les operateurs L_jk de la matrice dynamique.

    REFERENCE: equilibre_local_corrige.pdf, Section 5
    """
    # Termes diagonaux
    L_11 = props['C55'] * tau**2 - (props['C11']*delta1**2 + props['C66']*delta2**2)
    L_22 = props['C44'] * tau**2 - (props['C22']*delta2**2 + props['C66']*delta1**2)
    L_33 = props['C33'] * tau**2 - (props['C55']*delta1**2 + props['C44']*delta2**2)

    # Termes croises (symetriques)
    L_12 = L_21 = -(props['C12'] + props['C66']) * delta1 * delta2

    # Termes hors-plan (ANTISYMETRIQUES!)
    L_13 = +(props['C13'] + props['C55']) * delta1 * tau
    L_31 = -L_13  # Signe negatif!
    L_23 = +(props['C23'] + props['C44']) * delta2 * tau
    L_32 = -L_23  # Signe negatif!

    return {'L_11': L_11, 'L_12': L_12, ..., 'L_33': L_33}
```

Listing 11 – Opérateurs L_jk (core/mechanical_pdf.py)

**Termes Thermiques $Q_\alpha$**

$$Q_1 = (C_{11}\alpha_{11} + C_{12}\alpha_{22})\delta_1 \cdot T(x_3 = \bar{h}) \tag{37}$$
$$Q_2 = (C_{22}\alpha_{22} + C_{12}\alpha_{11})\delta_2 \cdot T(x_3 = \bar{h}) \tag{38}$$
$$Q_3 = (C_{13}\alpha_{11} + C_{23}\alpha_{22} + C_{33}\alpha_{33}) \cdot \left.\frac{dT}{dx_3}\right|_{x_3=\bar{h}} \tag{39}$$

```python
def compute_Q_thermal_vector(delta1, delta2, T, dT_dx3, alpha_coeffs, props):
    """
    Calcule le vecteur de sollicitation thermique Q = [Q1, Q2, Q3].

    REFERENCE: equilibre_local_corrige.pdf, Section 7 (Eq. 266-269)
    """
    Q1 = (props['C11']*alpha['alpha_1'] + props['C12']*alpha['alpha_2']) * delta1 * T
    Q2 = (props['C22']*alpha['alpha_2'] + props['C12']*alpha['alpha_1']) * delta2 * T
    Q3 = (props['C13']*alpha['alpha_1'] + props['C23']*alpha['alpha_2']
          + props['C33']*alpha['alpha_3']) * dT_dx3
    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 :

$$\boxed{\Gamma(\tau) = \begin{pmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{pmatrix}} \tag{40}$$

**Propriétés :**

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

```python
def get_Gamma_matrix(tau, delta1, delta2, props):
    """
    Construit la matrice dynamique Gamma(tau) 3x3.

    REFERENCE: equilibre_local_corrige.pdf, Section 6 (Eq. 199-208)
    """
    L = compute_L_operators(tau, delta1, delta2, props)
    return np.array([
        [L['L_11'], L['L_12'], L['L_13']],
        [L['L_21'], L['L_22'], L['L_23']],
        [L['L_31'], L['L_32'], L['L_33']]
    ], dtype=complex)
```

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

### 5.6.4 Section 7 : Assemblage Matriciel $9 \times 9$

**Matrice Bloc-Diagonale $K_{Dyn}$**

La matrice globale $9 \times 9$ est bloc-diagonale :

$$\boxed{\mathbb{K}_{Dyn} = \begin{pmatrix} \Gamma(\tau_1) & 0 & 0 \\ 0 & \Gamma(\tau_2) & 0 \\ 0 & 0 & \Gamma(\tau_3) \end{pmatrix}_{9\times9}} \tag{41}$$

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

```python
def assemble_K_dyn_9x9(tau_roots, delta1, delta2, props):
    """
    Assemble la matrice dynamique bloc-diagonale 9x9.

    REFERENCE: equilibre_local_corrige.pdf, Section 7 (Eq. 220-235)
    """
    K_dyn = np.zeros((9, 9), dtype=complex)

    for r in range(3):
        Gamma_r = get_Gamma_matrix(tau_roots[r], delta1, delta2, props)
        start, end = 3*r, 3*(r+1)
        K_dyn[start:end, start:end] = Gamma_r

    return K_dyn
```

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

**Système Final :**

$$\mathbb{K}_{Dyn} \cdot \mathcal{A} = \mathcal{F}_{Th} \tag{42}$$

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)

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

```python
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 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 $C_{ij}$, $\alpha_{ij}$ |
| Étape 5 : Ansatz | `get_Gamma_matrix()` | mechanical_pdf.py | Construction $\Gamma(\tau)$ avec $L_{jk}$ |

| Étape PDF | Fonction Python | Fichier | Description |
|---|---|---|---|
| Étape 6 : Valeurs propres | `solve_characteristic_polynomial()` | mechanical_pdf.py | Résolution $\det(\Gamma) = 0$ |
| Étape 6 : Sélection $\tau_r$ | – | mechanical_pdf.py | 3 racines avec $\mathrm{Re}(\tau) < 0$ |
| Étape 7 : Termes $\mathcal{Q}_\alpha$ | `compute_Q_thermal_vector()` | mechanical_pdf.py | Sollicitation thermique |
| Étape 7 : Assemblage $9 \times 9$ | `assemble_K_dyn_9x9()` | mechanical_pdf.py | $\mathbb{K}_{Dyn}$ bloc-diagonal |
| Étape 8 : Résolution | `solve_amplitude_system()` | mechanical_pdf.py | $\mathbb{K}_{Dyn} \cdot \mathcal{A} = \mathcal{F}_{Th}$ |
| Étape 8 : Système global | `solve_multilayer()` | mechanical.py | $K_{glob} \cdot C = F$ |
| 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 $\beta_i$

Les coefficients $\beta_i$ relient les coefficients de dilatation thermique $\alpha_j$ aux contraintes induites via le tenseur de rigidité :

**Calcul des Coefficients Beta**

$$\beta_i = \sum_{j=1}^{3} C_{ij}\alpha_j = C_{i1}\alpha_1 + C_{i2}\alpha_2 + C_{i3}\alpha_3 \tag{43}$$

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

$$\beta_1 = (C_{11} + C_{12} + C_{13})\alpha \tag{44}$$
$$\beta_2 = (C_{12} + C_{22} + C_{23})\alpha \tag{45}$$
$$\beta_3 = (C_{13} + C_{23} + C_{33})\alpha \tag{46}$$

```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 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**

$$\boxed{\mathbf{F}_{th} = \hat{T} \begin{pmatrix} \beta_1 \delta_1 \\ \beta_2 \delta_2 \\ \beta_3 \lambda_{th} \end{pmatrix}} \tag{47}$$

où $\hat{T}$ est l'amplitude de la perturbation de température et $\lambda_{th}$ 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

$$D_r \cdot K_{glob} \cdot D_c \cdot \mathbf{y} = D_r \cdot \mathbf{F} \tag{48}$$

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

$$D_r[i,i] = \frac{1}{\max_j |K[i,j]|} \tag{49}$$

$$D_c[j,j] = \frac{1}{\max_i |K_{scaled}[i,j]|} \tag{50}$$

### 9.2 Régularisation de Tikhonov

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

$$K = U \Sigma V^H \tag{51}$$

La solution régularisée est :

$$\boxed{\mathbf{x}_{reg} = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} \cdot \frac{\mathbf{u}_i^H \mathbf{b}}{\sigma_i} \cdot \mathbf{v}_i} \tag{52}$$

où $\lambda$ 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}
```

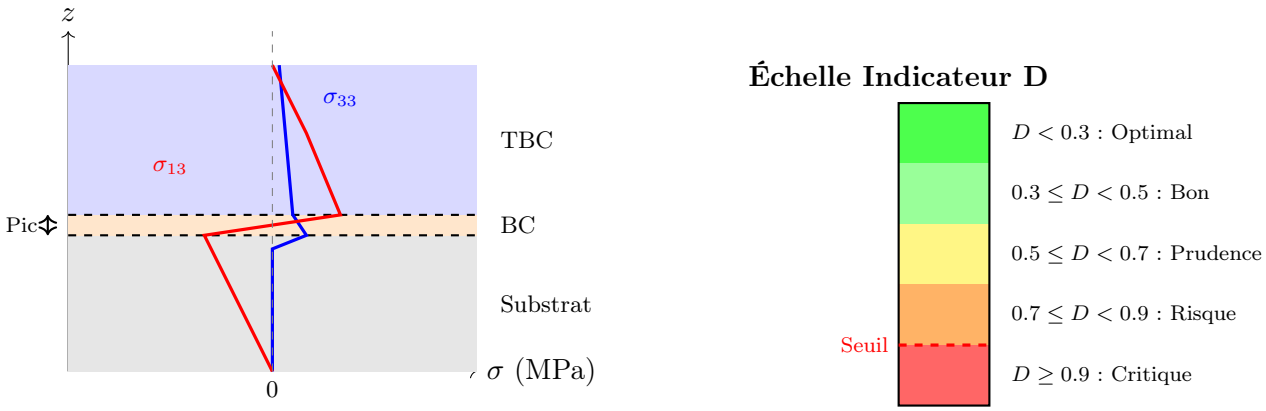
Listing 18 – Régularisation Tikhonov (core/mechanical.py lignes 846-989)

# 10 Interprétation Physique des Résultats

## 10.1 Signification des Contraintes Transverses

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

**Profils de Contraintes Typiques dans une Structure TBC**



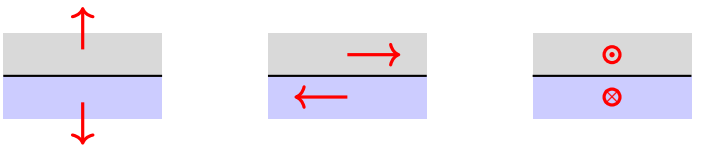

## 10.2 Zones Critiques Typiques

- **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 $\sigma_{max}$ | 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



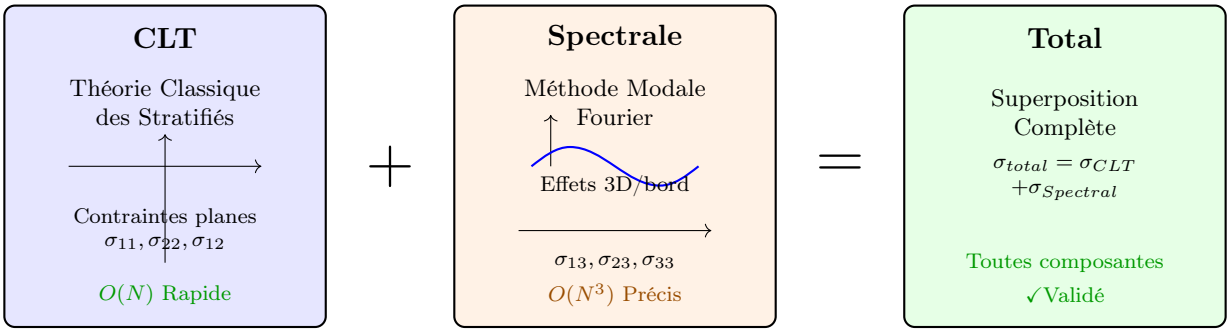
**Mode I**
Ouverture
$\sigma_{33} > 0$

**Mode II**
Cisaillement plan
$\sigma_{13}$

**Mode III**
Cisaillement antiplan
$\sigma_{23}$

## 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 $\sigma_{11}, \sigma_{22}$ |
| 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 :

$$\sigma_{total} = \sigma_{CLT} + \sigma_{Spectral} \tag{53}$$

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 |
| $\alpha$ à RT ($K^{-1}$) | $4.95 \times 10^{-6}$ | $12 \times 10^{-6}$ (moyenne T) |
| $\alpha$ à 1198K ($K^{-1}$) | $14.68 \times 10^{-6}$ | – |

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

# 13 Annexe A : Équilibre Local et Solution Détaillée

Cette annexe reprend la dérivation complète de la méthode de résolution selon le document `equilibre_local_co`

## 13.1 Équilibre Local en Formulation Forte

L'équilibre local est défini par :

$$\operatorname{div} \underline{\underline{\sigma}} = \vec{0} \tag{54}$$

En injectant la loi de comportement, on obtient le système d'équations différentielles suivant pour le milieu $(i)$.

### 13.1.1 Équations de Champ ($x_3$)

**Équation #1 : ($\alpha = 1, \beta = 2$)**

$$-\left(C^{(i)}_{1111}\delta_1^2 + C^{(i)}_{1212}\delta_2^2\right) U_1^{(i)}(x_3) + C^{(i)}_{1313}\frac{d^2U_1^{(i)}(x_3)}{dx_3^2} - \left(C^{(i)}_{1122} + C^{(i)}_{1212}\right)\delta_1\delta_2 U_2^{(i)}(x_3) + \left(C^{(i)}_{1133} + C^{(i)}_{1313}\right)\delta_1\frac{dU_3^{(i)}(x_3)}{dx_3} = \left(C^{(i)}_{1111}\alpha^{(i)}_{11} + C^{(i)}_{1122}\alpha^{(i)}_{22}\right)\delta_1 T(x_3 = \bar{h}^{(i)}) \tag{55}$$

**Équation #2 : ($\alpha = 2, \beta = 1$)**

$$-\left(C^{(i)}_{2222}\delta_2^2 + C^{(i)}_{1212}\delta_1^2\right) U_2^{(i)}(x_3) + C^{(i)}_{2323}\frac{d^2U_2^{(i)}(x_3)}{dx_3^2} - \left(C^{(i)}_{2211} + C^{(i)}_{1212}\right)\delta_2\delta_1 U_1^{(i)}(x_3) + \left(C^{(i)}_{2233} + C^{(i)}_{2323}\right)\delta_2\frac{dU_3^{(i)}(x_3)}{dx_3} = \left(C^{(i)}_{2222}\alpha^{(i)}_{22} + C^{(i)}_{2211}\alpha^{(i)}_{11}\right)\delta_2 T(x_3 = \bar{h}^{(i)}) \tag{56}$$

**Équation #3 : (Direction 3)**

$$\sum_{\gamma=1}^{2}\left[-(C^{(i)}_{\gamma\gamma 33} + C^{(i)}_{\gamma 3\gamma 3})\delta_\gamma\frac{dU_\gamma^{(i)}(x_3)}{dx_3} - C^{(i)}_{\gamma 3\gamma 3}\delta_\gamma^2 U_3^{(i)}(x_3)\right] + C^{(i)}_{3333}\frac{d^2U_3^{(i)}(x_3)}{dx_3^2} = \left(\sum_{\gamma=1}^{2} C^{(i)}_{\gamma\gamma 33}\alpha^{(i)}_{\gamma\gamma} + C^{(i)}_{3333}\alpha^{(i)}_{33}\right)\left.\frac{dT}{dx_3}\right|_{x_3=\bar{h}^{(i)}} \tag{57}$$

## 13.2 Forme de la Solution Générale

Pour $\alpha \in \{1, 2, 3\}$, la solution est recherchée sous forme de combinaison d'exponentielles :

$$U_1^{(i)}(x_3) = A_1^{1(i)}e^{\tau_1^{(i)}x_3} + A_1^{2(i)}e^{\tau_2^{(i)}x_3} + A_1^{3(i)}e^{\tau_3^{(i)}x_3} \tag{58}$$

$$U_2^{(i)}(x_3) = A_2^{1(i)}e^{\tau_1^{(i)}x_3} + A_2^{2(i)}e^{\tau_2^{(i)}x_3} + A_2^{3(i)}e^{\tau_3^{(i)}x_3} \tag{59}$$

$$U_3^{(i)}(x_3) = A_3^{1(i)}e^{\tau_1^{(i)}x_3} + A_3^{2(i)}e^{\tau_2^{(i)}x_3} + A_3^{3(i)}e^{\tau_3^{(i)}x_3} \tag{60}$$

Où :
— $A_\alpha^{r(i)}$ sont les amplitudes associées à chaque mode $r$ pour la direction $\alpha$.
— $\tau_r^{(i)}$ sont les valeurs propres du milieu $(i)$, issues de l'analyse du système homogène.

## 13.3 Dérivées Premières et Secondes

**Dérivées premières :**

$$\frac{dU_1^{(i)}(x_3)}{dx_3} = \sum_{r=1}^{3} \tau_r^{(i)} A_1^{r(i)} e^{\tau_r^{(i)} x_3} \tag{61}$$

$$\frac{dU_2^{(i)}(x_3)}{dx_3} = \sum_{r=1}^{3} \tau_r^{(i)} A_2^{r(i)} e^{\tau_r^{(i)} x_3} \tag{62}$$

$$\frac{dU_3^{(i)}(x_3)}{dx_3} = \sum_{r=1}^{3} \tau_r^{(i)} A_3^{r(i)} e^{\tau_r^{(i)} x_3} \tag{63}$$

**Dérivées secondes :**

$$\frac{d^2U_\alpha^{(i)}(x_3)}{dx_3^2} = \sum_{r=1}^{3} (\tau_r^{(i)})^2 A_\alpha^{r(i)} e^{\tau_r^{(i)} x_3} \tag{64}$$

## 13.4 Opérateurs $L_{jk}^{r(i)}$ de la Matrice Dynamique

**Termes diagonaux (couplage direct) :**

$$L_{11}^{r(i)} = C_{1313}^{(i)} (\tau_r^{(i)})^2 - \left( C_{1111}^{(i)} \delta_1^2 + C_{1212}^{(i)} \delta_2^2 \right) \tag{65}$$

$$L_{22}^{r(i)} = C_{2323}^{(i)} (\tau_r^{(i)})^2 - \left( C_{2222}^{(i)} \delta_2^2 + C_{1212}^{(i)} \delta_1^2 \right) \tag{66}$$

$$L_{33}^{r(i)} = C_{3333}^{(i)} (\tau_r^{(i)})^2 - \left( C_{1313}^{(i)} \delta_1^2 + C_{2323}^{(i)} \delta_2^2 \right) \tag{67}$$

**Termes croisés (couplage dans le plan) :**

$$L_{12}^{r(i)} = L_{21}^{r(i)} = -\left( C_{1122}^{(i)} + C_{1212}^{(i)} \right) \delta_1 \delta_2 \tag{68}$$

**Termes hors-plan (faisant intervenir $\tau_r^{(i)}$) — ANTISYMÉTRIQUES :**

$$L_{13}^{r(i)} = \left( C_{1133}^{(i)} + C_{1313}^{(i)} \right) \delta_1 \tau_r^{(i)} \tag{69}$$

$$L_{23}^{r(i)} = \left( C_{2233}^{(i)} + C_{2323}^{(i)} \right) \delta_2 \tau_r^{(i)} \tag{70}$$

$$L_{31}^{r(i)} = -\left( C_{1133}^{(i)} + C_{1313}^{(i)} \right) \delta_1 \tau_r^{(i)} \tag{71}$$

$$L_{32}^{r(i)} = -\left( C_{2233}^{(i)} + C_{2323}^{(i)} \right) \delta_2 \tau_r^{(i)} \tag{72}$$

**Antisymétrie Critique**

Les signes négatifs de $L_{31}$ et $L_{32}$ sont **essentiels** pour la physique correcte. Ils proviennent de l'équation d'équilibre en direction $x_3$.

### 13.5 Matrice $\Gamma(\tau)$ et Système Homogène

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

$$\boxed{\begin{bmatrix} \Gamma_{11}^{(i)} & \Gamma_{12}^{(i)} & \Gamma_{13}^{(i)} \\ \Gamma_{21}^{(i)} & \Gamma_{22}^{(i)} & \Gamma_{23}^{(i)} \\ \Gamma_{31}^{(i)} & \Gamma_{32}^{(i)} & \Gamma_{33}^{(i)} \end{bmatrix} \begin{pmatrix} A_1^{r(i)} \\ A_2^{r(i)} \\ A_3^{r(i)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}} \tag{73}$$

Avec les composantes :

$$\Gamma_{11}^{(i)} = C_{1313}^{(i)}(\tau_r^{(i)})^2 - (C_{1111}^{(i)}\delta_1^2 + C_{1212}^{(i)}\delta_2^2) \tag{74}$$
$$\Gamma_{22}^{(i)} = C_{2323}^{(i)}(\tau_r^{(i)})^2 - (C_{2222}^{(i)}\delta_2^2 + C_{1212}^{(i)}\delta_1^2) \tag{75}$$
$$\Gamma_{33}^{(i)} = C_{3333}^{(i)}(\tau_r^{(i)})^2 - (C_{1313}^{(i)}\delta_1^2 + C_{2323}^{(i)}\delta_2^2) \tag{76}$$
$$\Gamma_{12}^{(i)} = \Gamma_{21}^{(i)} = -(C_{1122}^{(i)} + C_{1212}^{(i)})\delta_1\delta_2 \tag{77}$$
$$\Gamma_{13}^{(i)} = (C_{1133}^{(i)} + C_{1313}^{(i)})\delta_1\tau_r^{(i)} \tag{78}$$
$$\Gamma_{23}^{(i)} = (C_{2233}^{(i)} + C_{2323}^{(i)})\delta_2\tau_r^{(i)} \tag{79}$$
$$\Gamma_{31}^{(i)} = -(C_{1133}^{(i)} + C_{1313}^{(i)})\delta_1\tau_r^{(i)} \tag{80}$$
$$\Gamma_{32}^{(i)} = -(C_{2233}^{(i)} + C_{2323}^{(i)})\delta_2\tau_r^{(i)} \tag{81}$$

### 13.6 Assemblage Complet du Système $9 \times 9$

Le système global pour une couche $(i)$ s'écrit :

$$\left[\mathbb{K}_{Dyn}^{(i)}\right]_{(9\times9)} \cdot \{\mathcal{A}^{(i)}\}_{(9\times1)} = \{\mathcal{F}_{Th}^{(i)}\}_{(9\times1)} \tag{82}$$

La matrice $\mathbb{K}_{Dyn}$ est **bloc-diagonale** :

$$\boxed{\mathbb{K}_{Dyn}^{(i)} = \begin{pmatrix} \Gamma(\tau_1) & 0 & 0 \\ 0 & \Gamma(\tau_2) & 0 \\ 0 & 0 & \Gamma(\tau_3) \end{pmatrix}_{9\times9}} \tag{83}$$

Le vecteur d'amplitudes $\mathcal{A}$ et le vecteur thermique $\mathcal{F}_{Th}$ :

$$\mathcal{A}^{(i)} = \begin{pmatrix} A_1^1 \\ A_2^1 \\ A_3^1 \\ A_1^2 \\ A_2^2 \\ A_3^2 \\ A_1^3 \\ A_2^3 \\ A_3^3 \end{pmatrix}^{(i)}, \quad \mathcal{F}_{Th}^{(i)} = \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_1 \\ Q_2 \\ Q_3 \\ Q_1 \\ Q_2 \\ Q_3 \end{pmatrix}^{(i)} \tag{84}$$

### 13.7 Termes de Sollicitation Thermique $\mathcal{Q}_\alpha$

$$\mathcal{Q}_1^{(i)} = \left(C_{1111}^{(i)}\alpha_{11}^{(i)} + C_{1122}^{(i)}\alpha_{22}^{(i)}\right)\delta_1\, T(x_3 = \bar{h}^{(i)}) \tag{85}$$
$$\mathcal{Q}_2^{(i)} = \left(C_{2222}^{(i)}\alpha_{22}^{(i)} + C_{2211}^{(i)}\alpha_{11}^{(i)}\right)\delta_2\, T(x_3 = \bar{h}^{(i)}) \tag{86}$$
$$\mathcal{Q}_3^{(i)} = \left(\sum_{\gamma=1}^{2} C_{\gamma\gamma33}^{(i)}\alpha_{\gamma\gamma}^{(i)} + C_{3333}^{(i)}\alpha_{33}^{(i)}\right)\left.\frac{dT}{dx_3}\right|_{x_3=\bar{h}^{(i)}} \tag{87}$$

# 14 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 $\alpha$ (épaisseur TBC) pour réduire les contraintes aux interfaces