ChatGPT

# Implementación del MITC4/MITC4+ en MITC4.py vs formulaciones teóricas

La implementación en *MITC4.py* sigue en esencia las formulaciones de Ko, Lee y Bathe para MITC4 y MITC4+ [1] [2]. A continuación examinamos detalladamente cada aspecto teórico y su reflejo en el código, señalando coincidencias o discrepancias.

## Interpolación geométrica y de desplazamientos

En los artículos originales (Ko et al., 2016, 2017) la geometría del elemento cuadrilateral se interpola mediante funciones de forma bilineales $h_i(r,s)$ (ec. (1)-(2) en [3]) y los desplazamientos de la línea media usan la misma interpolación (ec. (3) en [3]). En *MITC4.py*, la geometría inicial de los nodos se proyecta a un sistema de coordenadas local (con `compute_local_coordinates`), alineando el eje $x$ con el borde (0→1) y definiendo $y$ ortogonalmente. De este modo el elemento se trata como plano en 2D local, lo cual equivale al caso de elemento de líneas medias planas. El código usa funciones de forma bilineales estándar (véase `_get_N`), y para la transformación al espacio real usa la jacobiana local `J(r,s)`. Esto concuerda con la formulación canónica del MITC4/MITC4+: la interpolación isoparamétrica se aplica igual que en Bathe y Dvorkin (1984) [1]. La única diferencia sutil es que Ko et al. incluyen vectores directores y desplazamiento a través de espesor (ec. (1)-(2) en [3]), pero *MITC4.py* efectúa la formulación de submodelo delgado (Mindlin) degradando a desplazamientos rígidos normales al espesor (ρ = 0). En la práctica esto no contradice las hipótesis de MITC4, pues la deformación se calcula sobre la línea media.

## Cálculo de las deformaciones covariantes

Ko et al. definen las componentes covariantes de la deformación de Green-Lagrange en la línea media (ec. (4)-(5) en [4]). En el código, estas deformaciones se obtienen proyectando las derivadas espaciales de los desplazamientos locales sobre las bases covariantes $g_r, g_s$. En particular, el método `_evaluate_B_m_covariant(r,s)` construye la matriz de deformación de membrana covariante $[\tilde{e}_{rr}, \tilde{e}_{ss}, 2\tilde{e}_{rs}]$ respecto a los DOF $(u_i, v_i)$ en ejes locales (ec. (18-19) de Ko2016). El cálculo usa las componentes de $g_r, g_s$ en el marco local ortonormal $(e_1, e_2)$ y las derivadas $\partial N_i/\partial r$, $\partial N_i/\partial s$ estándar [5]. Esta formulación coincide con la ecuación (4) de Ko et al. (2017) para deformación covariante, traducida al elemento de membrana (ver [4] [6]). En resumen, **la implementación de las deformaciones lineales covariantes es correcta** conforme a la teoría de Ko et al.

## Formulación del elemento MITC4 estándar (sin “+”)

Para MITC4 clásico (bandera `use_mitc4plus=False`), el código construye la matriz $B_m$ de membrana mediante interpolación isoparamétrica plana: usa `_get_dH(r,s)` para obtener $\partial N_i/\partial x, \partial N_i/\partial y$ en coordenadas locales (resuelve J^-1 * dN/d(r,s) [7]), y luego arma $B_m$ 3×8 con: - $\varepsilon_{xx} = \partial u/\partial x$ (fila 0) y $\varepsilon_{yy} = \partial v/\partial y$ (fila 1) usando $\partial N_i$, - $\gamma_{xy} = \partial u/\partial y + \partial v/\partial x$ (fila 2) añadiendo ambas contribuciones [8].

Esta matriz coincide exactamente con el elemento cuadrilátero estándar (ecuaciones de deformación pequeñas), por lo que reproduce la formulación de Dvorkin-Bathe (MITC4) en su parte de membrana. Dvorkin & Bathe (1984) y Bathe (1996) también obtenían el mismo $B$ para el comportamiento membrana del elemento plano. Por tanto, **para $B_m$ de membrana lineal no hay discrepancia**. (Nótese que en Ko2017 se menciona que el elemento MITC4 estándar usa las ecuaciones (1)-(3) sin modificaciones [6], lo cual equivale a esta implementación isoparamétrica).

El código asume deformación infinitesimal en esta parte, pues la deformación membrana no incluye términos de segundo orden (se encarga en la parte no lineal, vista más abajo). Esta es la forma esperada: el elemento lineal MITC4 original usa deformación pequeña en membrana y asume campos lineales de desplazamiento.

## Interpolación MITC4 del campo de corte transversal

Para las deformaciones de cortante ($\gamma_{xz}$, $\gamma_{yz}$), MITC4 usa un campo asumido que evita bloqueo. Según Dvorkin & Bathe (1984) y Ko et al. (2017), se imponen campos constantes en cada borde medio de arista. El código implementa exactamente las ecuaciones (6) del MITC4:

$$\tilde{\varepsilon}_{rz} = \frac{1-s}{2}\varepsilon_{rz}|_A + \frac{1+s}{2}\varepsilon_{rz}|_C, \quad \tilde{\varepsilon}_{sz} = \frac{1-r}{2}\varepsilon_{sz}|_B + \frac{1+r}{2}\varepsilon_{sz}|_D,$$

con los puntos A=(0,-1),B=(-1,0),C=(0,1),D=(1,0) como en la Fig.2 de [9]. En el código, `_evaluate_covariant_shear_at_point` calcula en cada punto la contribución de $\varepsilon_{rz} = \partial w/\partial r + g_r \cdot \theta$ y $\varepsilon_{sz} = \partial w/\partial s + g_s \cdot \theta$ a los DOF $(w_i, \theta_{x,i}, \theta_{y,i})$ [10]. Luego, en `B_gamma(r,s)` se interpola linealmente según los factores $(1-s)/2,\ (1+s)/2,\ (1-r)/2,\ (1+r)/2$ tal como en la teoría [1] [10]. Finalmente se transforma de coordenadas paramétricas a local: si $J^{-1}$ es la inversa de la jacobiana local, el código aplica

$$[\gamma_{xz}, \gamma_{yz}]^T = J^{-T}[\tilde{\varepsilon}_{rz},\ \tilde{\varepsilon}_{sz}]^T,$$

es decir, $\gamma_{xz} = \frac{\partial r}{\partial x}\tilde{e}_{rz} + \frac{\partial s}{\partial x}\tilde{e}_{sz}$, $\gamma_{yz} = \frac{\partial r}{\partial y}\tilde{e}_{rz} + \frac{\partial s}{\partial y}\tilde{e}_{sz}$ [11]. Este paso de transformación es consistente con la formulación continua (típicamente $\gamma_{xz} = \partial w/\partial x + \ldots$). En suma, **la interpolación y transformación del cortante en el código coincide con la formulación MITC4 original** [1] [11].

## Campo de deformación membrana asumido (MITC4+)

La versión MITC4+ introduce un campo de deformación membrana asumido para corregir fallas en el patch test bajo mallas distorsionadas [2]. Ko et al. construyen este campo mezclando las deformaciones medidas en cinco puntos de muestreo (A,B,C,D en bordes y E en centro) con coeficientes de distorsión $a_A, \ldots, a_E$ dados en la ecuación (27) de [2]. *MITC4.py* implementa exactamente estas fórmulas (método `_B_m_8` con `use_mitc4plus=True`).

- Primero, se calculan los coeficientes $c_r, c_s$ y $d = c_r^2 + c_s^2 - 1$ a partir de los vectores característicos de la geometría (método `_compute_distortion_coefficients`). Luego los coeficientes $a_A, \ldots, a_E$ según Ko2016 ec. (27):

  $$a_A = \frac{c_r(c_r-1)}{2d},\ a_B = \frac{c_r(c_r+1)}{2d},\ a_C = \frac{c_s(c_s-1)}{2d},\ a_D = \frac{c_s(c_s+1)}{2d},\ a_E = \frac{2c_rc_s}{d},$$

  que coinciden con Ko et al. [2]. El código chequea caso $d \approx 0$ (malla no distorsionada) y en ese límite fija $a_i = 0$, recuperando el elemento plano.

- A partir de ahí, el código construye las deformaciones covariantes asumidas: para $\tilde{e}_{rr}$ (fila 0) implementa la eq. (27a) de Ko2016, usando los términos lineales y bilineales adecuados; lo mismo para $\tilde{e}_{ss}$ (27b); y para $\tilde{e}_{rs}$ (fila 2) usa (27c) [2]. En el código, por ejemplo:

```
B_cov_rr = (0.5*(1+s) - aA*(1 - s**2))*BA[0] + (0.5*(1-s) - aB*(1 -
s**2))*BB[0]
           + aC*(s**2-1)*BC[1] + aD*(s**2-1)*BD[1] + aE*(1 -
s**2)*0.5*BE[2]
```

  coincide con la forma dada en Ko et al. (ec.27a) tras notar que el código trabaja con deformaciones de ingeniería ($\gamma_{rs} = 2e_{rs}$), de ahí el factor 0.5 en la última componente. Todas estas expresiones emparejan las ec. (27a,b,c) de Ko et al. [2]. Finalmente, igual que en la teoría, se transforma a deformaciones cartesianas locales con la matriz $T$ dada en `_covariant_to_cartesian_strain_transform` [12].

Por tanto, **la implementación del campo de deformación MEMBRANA asumido en MITC4+ es fiel a las formulaciones originales**. En particular, los coeficientes $a_i$ y la mezcla de las matrices $B_{\text{cov}}$ en A–E concuerdan con Ko et al. (2016) [2]. El código incluso comenta explícitamente esta correspondencia.

## Formulación de Green–Lagrange y no linealidad geométrica

Para análisis no lineal el código usa formulación Lagrangiana total, con la deformación de Green–Lagrange. Calcula el gradiente de desplazamiento $H$ respecto a coordenadas locales (método `get_displacement_gradient`) y la deformación $E = \frac{1}{2}(H + H^T + H^T H)$ [13]. Esto es coherente con el TL descrito en Ko et al. (2017). La matriz $B_L$ (método `_compute_B_L`) corresponde a la parte lineal de la deformación ($\partial u/\partial x$, etc.), y $B_{NL}$ (`_compute_B_NL`) captura los términos cuadráticos en $H$ [14]. En el cálculo de fuerzas internas se integran tanto $B_L$ como $B_{NL}$ (suma de matrices) contra el esfuerzo de 2do P.K., como se ve en `compute_internal_forces` [15]. Esto es consistente con la teoría de Ko (2017), donde las contribuciones geométricas (stiffness σ) se obtienen de este modo. En resumen, **la formulación de campo de deformación no lineal coincide con GL estándar** y sigue las ecuaciones expuestas por Ko, Lee y Bathe en su trabajo no lineal (por ejemplo Ko2017 sección 3).

## Matrices constitutivas y ensamblaje de rigideces

El código usa las matrices constitutivas de membrana $C_m$, flexión $C_b$ y cortante $C_s$ como en la teoría (matrices de placas ortotrópicas o isotrópicas) [16] [17] [18]. Se integra con 2×2 puntos de Gauss las contribuciones membrana, flexión y cortante (métodos `k_m`, `k_b`). En `k_b` combina la rigidez de flexión ($B_\kappa^T C_b B_\kappa$) y de cortante ($B_\gamma^T C_s B_\gamma$) y luego añade una pequeña rigidez de taladro (“drilling”) para estabilizar las rotaciones alrededor de $z$ [19]. Aunque en los artículos originales no se discute el taladro, es una corrección numérica común y no afecta los modos de flexión-membrana fundamentales.

## Verificación de pruebas básicas y posibles discrepancias

Según Ko et al. (2016), el elemento MITC4+ pasa las pruebas de **parche de deformación uniforme, isotropía y modos de energía cero** [2] . Con la formulación implementada, el código debería cumplir estos requisitos *si se activa MITC4+*. En particular:

- **Test de parche (membrana).** Ko (2016) muestra que, gracias al campo asumido, incluso con malla distorsionada el elemento reproduce exacta deformación uniforme [2] . En el código esto sucede porque las combinaciones de $a_i$ están calculadas para satisfacer esas condiciones (ec. 21 de Ko2016). Si en cambio se deja `use_mitc4plus=False`, el elemento no incluye el campo de membrana corregido, por lo que en presencia de distorsión falla el parche (comportamiento esperado del MITC4 original). **No es un "error" del código**, sino del modelo empleado: MITC4+ debe estar activo para pasar el parche de membrana, tal como señala Ko et al. [2] .

- **Isotropía.** Dado que la formulación no introduce factores numéricos extra, la respuesta debe ser independiente de la orientación de la malla, como comprueba Ko2016 [2] . La implementación no altera las constantes constitutivas ni añade anisotropía artificial, por lo que se espera respuesta espacialmente isotrópica (salvo efectos de red discreta). No se observa ninguna formulación en el código que rompa la simetría del cuadrilátero; por tanto **coincide con lo reportado**.

- **Modos de energía cero (ej. deformaciones rígidas).** El elemento debe excluir modos no físicos. El código estabiliza los giros en z con rigidez casi nula, previniendo modos de energía cero asociados. Las deformaciones rígidas genuinas (traslación/rotación) anulan $B$ por construcción, como es correcto. No hemos encontrado errores lógicos que introduzcan modos extra: las matrices $B_m, B_\gamma, B_\kappa$ están bien definidas en cada DOF, por lo que la nulidad para rigideces se cumple.

En síntesis, **la implementación respeta las propiedades básicas teóricas**. No obstante, conviene señalar algunos puntos de mejora/precisión:

1. **Convención de coordenadas y orden de nodos.** El código asume nodos ordenados (0=esq. inf-izq, 1=inf-der, 2=sup-der, 3=sup-izq) para construir el sistema local [20] . Si el usuario provee otro orden, habría que redefinirlo para mantener coherencia. Un posible refactor es validar o reordenar internamente los nodos según esta convención.

2. **Claridad de transformaciones.** Se usan dos marcos locales: el definido por el eje (0→1)-(0→3) para desplazamientos, y el marco tangente $(e_1, e_2)$ en cada Gauss para deformaciones covariantes. Aunque es correcto, el código podría documentar más explícitamente estas distinciones (por ejemplo, nombrar variables `g_r_global` vs `g_r_local`, etc.) para mejorar legibilidad.

3. **Factor de normalización en e_rs**. Ko (2016) usa la deformación tensorial $e_{rs}$ en ec. (27), mientras el código trabaja con la deformación de ingeniería $\gamma_{rs} = 2e_{rs}$. El código añade explícitamente el factor 0.5 donde corresponde (comentario en [21] ). Esto es correcto pero delicado: cualquier refactor debe preservar este ajuste.

4. **Eficiencia/Caching.** El código ya aprovecha caching de jacobianos y derivadas (*dH_cache*, *J_cache*), lo cual es bueno. Quizá podría unificar en una sola rutina la obtención de $B_m$ covariante

en A–E (ya hace caching de cada $B^A_{\text{cov}}$ en `__init__`). También podría precalcular el Jacobiano inverso para B_gamma, aunque con pocos pts Gauss no es crítico.

5. **Puesta en obra de la formulación de energía.** En `compute_internal_forces` se separan contribuciones membrana, flexión y corte; esto coincide con la formulación de placas-suelo de Ko/Bathe. Podría aclararse en comentarios cómo se integran los resultados (especialmente la extracción de DOFs membrana).

En conclusión, **no se detectaron errores fundamentales** en las fórmulas clave: el código implementa fielmente las interpolaciones, tensores de deformación, matrices $B$, formulación de Green–Lagrange y campos de deformación asumidos según los artículos de Ko, Lee y Bathe [1] [2]. Para garantizar las propiedades teóricas (parche, isotropía, modos cero) basta usar `use_mitc4plus=True`. Por claridad y mantenimiento, se sugieren comentarios adicionales (especialmente distinguiendo sistemas de referencia) y validación del orden de nodos, pero la estructura algorítmica esencial ya cumple los modelos originales.

**Referencias:** Las fórmulas del elemento MITC4 se detallan en Ko et al. 2017 [1]; el refinamiento MITC4+ y sus coeficientes de distorsión en Ko et al. 2016 [2]. Todos los elementos implementados en el código están basados en estas ecuaciones.

---

[1] [2] [3] [4] [6] [9] A new MITC4+ shell element
http://web.mit.edu/kjb/www/Principal_Publications/A_new_MITC4+_shell_element.pdf

[5] [7] [8] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] MITC4.py
file://file_00000000a824720eb47a0a27d9accd73