opbasis

automated construction of minimal operator bases for lattice QCD

EPJC 86 (2026) 4, 427 [arXiv:2511.00165]

https://github.com/nikolai-husung/opbasis

Nikolai Husung
27 July 2026

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES
Cofinanciado por la Unión Europea
AGENCIA ESTATAL DE INVESTIGACIÓN
CERN

Yet another package?

Most tools are intended for actions of continuum SMEFT and similar EFTs ☹

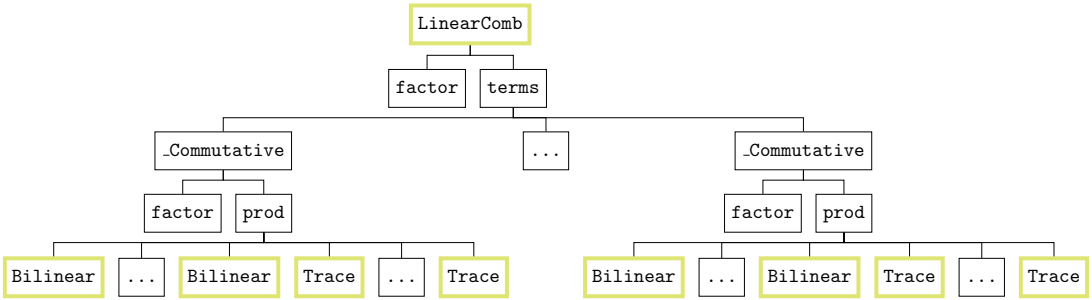
We want local fields with non-trivial transformation properties and

- Euclidean signature
- Spurions
- Euclidean reflections
- Charge conjugation

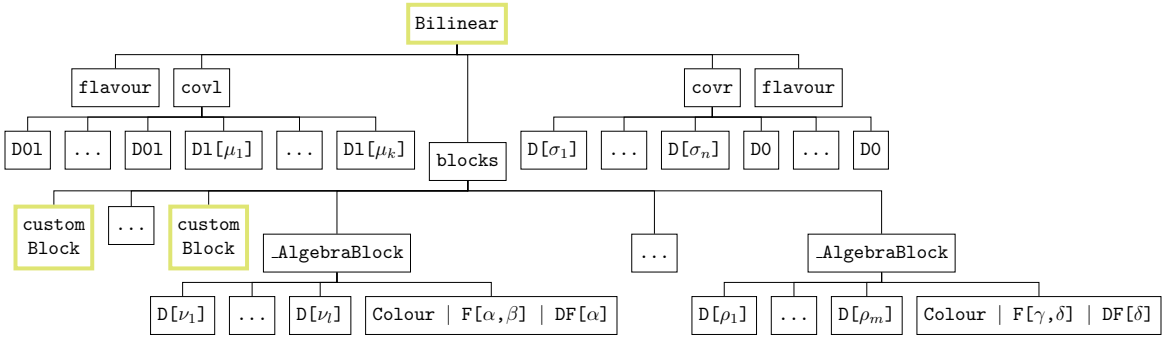
- Broken O(4) symmetry (typically H_4)
- Potentially discrete flavour symmetries
- ...
  - No real need for "trivial" continuous symmetries

⇝ Symanzik EFT operator bases and more

Rough structure



Rough structure



Custom `Block` implementation allows to introduce new building blocks, e.g., twisted mass parameter, Taste-space matrices, Boost-momentum, ...

(Typical) workflow

1) Define your model
   Model file + declaration of transformations (+ implementation of custom ones)

2a) Work with templates.
   1. Auto-generate templates (or by hand ϟ) for a specific mass-dimension.
   2. Turn templates into all possible variants (e.g., indices, Gamma structures, ...)

2b) Start from custom ansatz. ϟ ⇐ Use external input. (optional)

3) Construct valid operators according to requested transformation properties.

4) Reduction to minimal basis by linear independence.
   ⇝ Hint: Adding EOM-vanishing operators first allows to identify on-shell basis.

Example: Pure-gauge energy-momentum-tensor $\hat{T}_{01}$ to $O(a^2)$

$T_{01} = \mathrm{tr}(F_{0\mu}F_{1\mu})$

1) Define your model

models/EMT01.in

```
1  Transf: default
2  Discrete: P --++
3  Discrete: C +
4  Discrete: H4 -xxxx+
```

Use standard lattice transformations/default.py.

Transforms odd in (0,1)-plane and even in (2,3).

Example: Pure-gauge energy-momentum-tensor $\hat{T}_{01}$ to $O(a^2)$

$T_{01} = \mathrm{tr}(F_{0\mu}F_{1\mu})$

1) Define your model

models/EMT01.in

```
1 Transf: default
2 Discrete: P --++
3 Discrete: C +
4 Discrete: H4 -xxxx+
```

Use standard lattice transformations/default.py.

Transforms odd in (0,1)-plane and even in (2,3).

findEMT.py

```
5 name  = sys.argv[1]
6 md    = int(sys.argv[2])
7 model = opb.Model.read("models/%s.in"%name)
8 model.apply()
```

Example: Pure-gauge energy-momentum-tensor $\hat{T}_{01}$ to $\mathrm{O}(a^2)$

2a) Auto-generate templates

Choose desired canonical mass-dimension.

findEMT.py

```
10  templates = opb.getTemplates(md, None)
11  # Sort templates: EOM-vanishing > #(total der.) > #F(in Trace)
12  templates = list(sorted(templates, reverse=True,
13     key=lambda x: 7**2*x["DF"]+7*x["d"]+x["F"]))
```

Example: Pure-gauge energy-momentum-tensor $\hat{T}_{01}$ to $\mathrm{O}(a^2)$

2a) Auto-generate templates

Choose desired canonical mass-dimension.

findEMT.py

```
10  templates = opb.getTemplates( md , None)
11  # Sort templates: EOM-vanishing > #(total der.) > #F(in Trace)
12  templates = list(sorted(templates, reverse=True,
13     key=lambda x: 7**2*x["DF"]+7*x["d"]+x["F"]))
```

3) Construct valid operators

findEMT.py

```
16  bases = list()
17  for template in templates:
18     basis = opb.overcompleteBasis(str(template), model)
19     if len(basis)>0:
20        bases.append(basis)
```

Example: Pure-gauge energy-momentum-tensor $\hat{T}_{01}$ to $\mathrm{O}(a^2)$

$T_{01} = \mathrm{tr}(F_{0\mu}F_{1\mu})$

4) Reduction to minimal basis

Require full expression of EOM to unmask.

findEMT.py

```
30  bases = opb.findMinBases(bases, gEOM, None, model)
```

⇒ Minimal on-shell basis of 13 operators describing $\mathrm{O}(a^2)$ lattice artifacts of $\hat{T}_{01}$, e.g.,

$$(T_{01})_1^{(2)} = \partial_0\partial_1 \,\mathrm{tr}(F_{01}F_{01})$$

Recent addition: (Irreducible) representations

- Group-theoretic approach based on character tables to project to desired irrep.
  $\rightsquigarrow$ Requires elements of conjugacy classes mapped to chains of transformations.
- Row-projectors for multi-dimensional irreps can be deferred from a projected seed and reused.
- Can be combined with earlier workflow, e.g.,
  - Use initial operator basis as seed.
  - Enforce definite sign under charge.
- Boosted multi-meson interpolators can be build using custom `Boost` implementation, see [Husung, 2026].

Summary

Example provided alongside the slides.

- Original motivation: Automated identification of Symanzik EFT operator bases including unconventional lattice setups.
- Custom `Block` implementation covers non-trivial lattice transformation behaviour (flavour, reflections, ...) and involved irrep construction (isospin, momenta, ...).
- Operator bases also identify renormalisation patterns — additive or non-canonical operator mixing due to relaxed symmetries.
- Can be used to build suitable interpolating operators, see also talk by Juan Andrés Urrea-Niño earlier today and [Urrea-Niño et al., 2026].
- Not optimised for speed: Only needs to be run once per mass-dimension and requested transformation behaviour. Overcomplete basis can be stored and reused!
- Rather exhaustive documentation available: `https://nikolai-husung.github.io/opbasis/`

Possible plans for the future

- Currently only coefficients $c \in \mathbb{Q} + i\mathbb{Q}$ supported $\rightsquigarrow$ SymPy?
- Multiple or Abelian gauge groups (e.g. QCD+QED) will require some redesign.
- 6-quark operators would require generalised Fierz identities for a minimal basis.

References

N. Husung. opbasis: a Python package to derive minimal operator bases. *Eur. Phys. J. C*, 86(4):427, 2026.

J. A. Urrea-Niño, F. Knechtli, T. Korzec, and M. Peardon. Charmonium-glueball spectroscopy with improved hadron creation operators. *Phys. Rev. D*, 114(1): 014509, 2026.