Users' Guide for LCT Code

Nikhil Anand,[a] **A. Liam Fitzpatrick,**[b] **Emanuel Katz,**[b] **Zuhair U. Khandker,**[b,c] **Matthew T. Walters,**[d,e] **Yuan Xin**[b]

[a] *Department of Physics, McGill University, Montréal, QC H3A 2T8, Canada*

[b] *Department of Physics, Boston University, Boston, MA 02215, USA*

[c] *Department of Physics, University of Illinois, Urbana, IL 61801, USA*

[d] *Theoretical Physics Department, CERN, 1211 Geneva 23, Switzerland*

[e] *Institute of Physics, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland*

Contents

1 Overview

Here we provide an overview of the packages provided in the code, as well as demonstrations for how to run each of them and how to reproduce most of the numeric results in the paper. The interactions covered by the packages are

- ϕ^n for $n = 2, 3, 4$ in a theory with a scalar as well as a theory with a scalar and a fermion,
- The fermion mass term $\sim \psi \frac{1}{\partial} \psi$ in a theory with a fermion as well as a theory with a scalar and a fermion,
- A Yukawa interaction $\sim \phi\psi\chi$ in a theory with a scalar and a fermion,
- The supercharges Q_+, Q_- in a theory with a scalar and a fermion, for a ϕ^2 and ϕ^3 superpotential.

In subsection 3.2, we also describe a package and its demonstration for computing matrix elements in 2d QCD at finite N_c in the absence of a quark mass term.

2 Packages

To use any of the packages listed below, first make sure that the package is in the correct directory (e.g. `NotebookDirectory[]`). Then, import the package with the command

```
In[1]:= << "./PackageName.wl"
```

2.1 `Basis-Scalar.wl`

The package `Basis-Scalar.wl` contains the routines needed to compute the scalar basis. The main function the user can call is `computeBosonBasis[`$\Delta_{\max}$`, prec]` which computes the scalar basis up to the maximum scaling dimension $\Delta_{\max}$, using numerical precision `prec` when orthogonalizing states. The basis is saved to the public variable `basisBoson`, where the basis states are orthonormalized numerically, as well as the public variable `basisBosonPre`, where the basis states are not orthonormalized. The advantage of `basisBosonPre` is that prior to orthonormalization, the primaries can be written as sums over monomials with rational coefficients and therefore it has infinite precision, allowing one to choose the numerical precision of the orthonormalization at a later time.

The organization of `basisBoson` is, in particular,

- `basisBoson[[1,deg+1,n]]` returns the basis states for particle number `n` and degree `deg` as a matrix whose rows correspond to different basis states, whose columns are monomials ordered according to `monomialsBoson[n,deg]`, and whose entries tell us the expansion of the basis states in terms of monomials.

For example, for `n=3` and `deg=2`, we have

```
In[2]:= n=3; deg=2;
        monomialsBoson[n,deg]
        basisBoson[[1,deg+1,n]]

Out[2]= {{3,1,1}, {2,2,1}}
        {{0.755929,-0.654654}}
```

which tells us that there is a single primary operator at this level given by

$$|\mathcal{O}\rangle = 0.755929\,|\partial^3\phi(\partial\phi)^2\rangle_{\rm RQ} - 0.654654\,|(\partial^2\phi)^2\partial\phi\rangle_{\rm RQ}\,. \tag{2.1}$$

The normalization convention for monomials used in `Basis-Scalar.wl` is

$$|\partial^{\boldsymbol{k}}\phi\rangle_{\rm RQ} \equiv \frac{1}{\mathcal{N}_{\boldsymbol{k}}\,\|\boldsymbol{k}\|}\partial^{\boldsymbol{k}}\phi(0)\,|{\rm vac}\rangle \quad \Rightarrow \quad \Big(|\partial^{\boldsymbol{k}'}\phi\rangle_{\rm RQ}\Big)^\dagger\cdot|\partial^{\boldsymbol{k}}\phi\rangle_{\rm RQ} = \delta_{\boldsymbol{k},\boldsymbol{k}'}, \tag{2.2}$$

where we are using the notation introduced in section 7.2 of the paper and † is Hermitian conjugation in the standard radial quantization sense. If needed, we can easily convert (2.1) back to a position space operator using (2.2). Letting $\boldsymbol{k}_1 = (3,1,1)$ and $\boldsymbol{k}_2 = (2,2,1)$, we have

$$\mathcal{O}(x) = \frac{0.755929}{\mathcal{N}_{\boldsymbol{k}_1}\,\|\boldsymbol{k}_1\|}\partial^{\boldsymbol{k}_1}\phi(x) - \frac{0.654654}{\mathcal{N}_{\boldsymbol{k}_2}\,\|\boldsymbol{k}_2\|}\partial^{\boldsymbol{k}_2}\phi(x) \propto 6\partial^{\boldsymbol{k}_1}\phi(x) - 9\partial^{\boldsymbol{k}_2}\phi(x). \tag{2.3}$$

This is the operator $\mathcal{O}_{(2,0)}$ (or equivalently $\mathcal{O}_{(0,2)}$) appearing in Table 2 of the paper.

Note: Much as the primary states $\mathcal{O}(0)|{\rm vac}\rangle$ in radial quantization are different from the primary states $|\mathcal{O},p\rangle$ in momentum space, the monomial states $|\partial^{\boldsymbol{k}}\phi\rangle_{\rm RQ}$ in radial quantization are different from the monomial states $|\partial^{\boldsymbol{k}}\phi,p\rangle$ in momentum space, with different inner product matrices (for instance $\big(|\partial^{\boldsymbol{k}'}\phi,p'\rangle\big)^\dagger\cdot|\partial^{\boldsymbol{k}}\phi,p\rangle$ does not vanish for $\boldsymbol{k}\neq\boldsymbol{k}'$, in contrast with (2.2)) and different meanings for Hermitian conjugation. Consequently, the relative coefficients of different monomials in the decomposition of a

primary operator $\mathcal{O}$ depend on which representation of the operator is used:

$$\begin{aligned}\mathcal{O}(x) &= \sum_{\boldsymbol{k}} C^{\mathcal{O}}_{\boldsymbol{k}} \partial^{\boldsymbol{k}} \phi(x), \\ |\mathcal{O}, p\rangle &= \sum_{\boldsymbol{k}} \frac{C^{\mathcal{O}}_{\boldsymbol{k}} N_k}{N_{\mathcal{O}}} |\partial^{\boldsymbol{k}} \phi, p\rangle \equiv \sum_{\boldsymbol{k}} \widehat{C}^{\mathcal{O}}_{k} |\partial^{\boldsymbol{k}} \phi, p\rangle, \\ \mathcal{O}(0)|\text{vac}\rangle &= \sum_{\boldsymbol{k}} C^{\mathcal{O}}_{\boldsymbol{k}} \mathcal{N}_{\boldsymbol{k}} \, \|\boldsymbol{k}\| \, |\partial^{\boldsymbol{k}} \phi\rangle_{\text{RQ}} \,. \end{aligned} \tag{2.4}$$

The reader should keep this in mind if comparing the above decomposition of $\mathcal{O}_{(2,0)}$ to that in section 4.2 (for instance, in Table 5) of the paper, where momentum-space monomials are used.

2.2 `Basis-Fermion.wl`

The package `Basis-Fermion.wl` contains the routines needed to compute the fermion basis. The main function the user can call is `computeFermionBasis[`Δ_{max}`, prec]` or `computeFermionBasis[`Δ_{max}`, prec, SUSY->True]`, which computes the fermion basis up to Δ_{max}, using numerical precision `prec` when orthogonalizing states. The meaning of Δ_{max} depends on the option `SUSY` (default `SUSY->False`):

- For option `SUSY->False`, Δ_{max} = `deg+3/2n` is the maximum scaling dimension, where `n` is the particle number and `deg` is the degree
- For option `SUSY->True`, we define Δ_{max} = `deg+n`, so that the truncation is compatible with supersymmery.

The basis is saved to the public variable `basisFermion`. The organization of `basisFermion` is analogous to that of its scalar counterpart `basisBoson`. In particular,

- `basisFermion[[1,deg+1,n]]` returns the basis states for particle number `n` and degree `deg` as a matrix whose rows correspond to different basis states, whose columns are monomials ordered according to `monomialsFermion[n,deg]`, and whose entries tell us the expansion of the basis states in terms of monomials.

For example, for `n=2` and `deg=3`, we have

```
In[3]:= n=2; deg=3;
        monomialsFermion[n,deg]
        basisFermion[[1,deg+1,n]]

Out[3]= {{4,1},{3,2}}
        {{-0.408248,0.912871}}
```

which tells us that there is a single primary operator at this level given by

$$|\mathcal{O}\rangle = -0.408248\,|\partial^4\psi\partial^2\psi\rangle_{\rm RQ} + 0.912871\,|\partial^3\psi\partial^2\psi\rangle_{\rm RQ}\,.$$

Just like in `Basis-Scalar.wl`, monomials in `Basis-Fermion.wl` are normalized according to (2.2). Taking this into account, we can convert the operator above to position space if desired, as was done for the scalar example in (2.3). One finds that the operator above corresponds to the operator called $\mathcal{O}_{(3)}$ in Table 11 of the paper.

The routines in `Basis-Fermion.wl` make frequent use of a correspondence between fermion and scalar monomials. As discussed in section 5.2 of the paper, there is a one-to-one mapping between fermion monomials of degree `deg` and scalar monomials of degree `bDeg`, where

$$\texttt{bDeg} = \texttt{deg} - \texttt{n}(\texttt{n}-1)/2$$

2.3 `Basis-Mixon.wl`

The package `Basis-Mixon.wl` contains the routines needed to compute the mixon basis. The main function the user can call is
`computeMixonBasis[`$\Delta_{\rm max}$`,basisBoson,basisFermion]`
or `computeMixonBasis[`$\Delta_{\rm max}$`,basisBoson,basisFermion,SUSY->True]`,
which computes the mixon basis up to $\Delta_{\rm max}$. The meaning of $\Delta_{\rm max}$ depends on the option `SUSY` (default `SUSY->False`), and is the same as described above for `Basis-Fermion.wl`.

The structure of the mixon basis is such that `basisMixon[[nB+1,nF+1]]` contains the set of levels in that particle number sector. Each level corresponds to a certain combination of {`degB,degF,l`}, where

- `degB` is the degree of the scalar primary state $\mathcal{O}_B$ that the mixon primary is built upon.
- `degF` is that of the fermion primary state $\mathcal{O}_F$
- `l` is the number of alternating derivative taken in the construction $\mathcal{O}_B\overleftrightarrow{\partial}^{\ell}\mathcal{O}_F$.

For example, in the sector `(nB=1, nF=0)` there is only one primary level, containing only one primary state.

```
In[4]:= nB=1; nF=0;
        basisMixon[[nB+1,nF+1]]

Out[4]= {<|"nB"->1, "degB"->0, "nF"->0, "degF"->0, "l"->0,
        "states"->{{{{1.}}}}|>}
```

For a non-trivial set

```
In[5]:= nB=2; nF=3;
        {#nB,#degB,#nF,#degF,#l}&/@basisMixon[[nB+1,nF+1]]//Column

Out[5]= {2,0,3,3,0}
        {2,0,3,3,1}
        {2,0,3,3,2}
        {2,0,3,5,0}
        {2,2,3,3,0}
```

For each level specified by {nB, degB, nF, degF, l}, there can be a number of states, each specified by (l+1) blocks of coeffcients where each block is a matrix of coefficients multiplying the matrix of mixon monomials given by monomialsBoson[nB, degB + m] $\otimes$ monomialsFermion[nF, degF + l - m] where m = 0,1,...,l. For example:

```
In[6]:= nB=2; nF=2;
        example=basisMixon[[nB+1,nF+1,5]]

Out[6]= <|"nB"->2, "degB"->0, "nF"->2, "degF"->1, "l"->4,
     "states"->{{{{-0.0778817,-0.0684043,-0.0263288}}, {{0.289617,
          0.20479}}, {{-0.434425,-0.194281}, {-0.354707,-0.15863}},
    {{0.388562}, {0.475889}}, {{-0.175534}, {-0.222035}, {-0.166527}}}}|>
```

there is only one state in the level, the state is specified by coefficients

```
In[7]:= example["states"][[1]] // Column

Out[7]= {{-0.0778817,-0.0684043,-0.0263288}}
        {{0.289617,0.20479}}
        {{-0.434425,-0.194281},{-0.354707,-0.15863}}
        {{0.388562},{0.475889}}
        {{-0.175534},{-0.222035},{-0.166527}}
```

multiplying the monomials

```
In[8]:= With[{nB=2,nF=2,degB=0,degF=1,l=4},
          Table[Outer[
            List[Flatten[{#1,#2}]]&,
            monomialsBoson[nB,degB+m],
```

```
    monomialsFermion[nF,degF+l-m],1],
  {m,0,l}]
] // Column
```

Out[8]=
```
{{{{1,1,6,1}},{{1,1,5,2}},{{1,1,4,3}}}}
{{{{2,1,5,1}},{{2,1,4,2}}}}
{{{{3,1,4,1}},{{3,1,3,2}}},{{{2,2,4,1}},{{2,2,3,2}}}}
{{{{4,1,3,1}}},{{{3,2,3,1}}}}
{{{{5,1,2,1}}},{{{4,2,2,1}}},{{{3,3,2,1}}}}
```

which represents the state

$$
\begin{aligned}
|\mathcal{O}\rangle = \\
&- 0.0778817\,|\partial\phi\partial\phi\partial^6\psi\partial\psi\rangle - 0.0684043\,|\partial\phi\partial\phi\partial^5\psi\partial^2\psi\rangle - 0.0263288\,|\partial\phi\partial\phi\partial^4\psi\partial^3\psi\rangle \\
&+ 0.289617\,|\partial^2\phi\partial\phi\partial^5\psi\partial\psi\rangle + 0.20479\,|\partial^2\phi\partial\phi\partial^4\psi\partial^2\psi\rangle \\
&- 0.434425\,|\partial^3\phi\partial\phi\partial^4\psi\partial\psi\rangle - 0.194281\,|\partial^3\phi\partial\phi\partial^3\psi\partial^2\psi\rangle \\
&- 0.354707\,|\partial^2\phi\partial^2\phi\partial^4\psi\partial\psi\rangle - 0.15863\,|\partial^2\phi\partial^2\phi\partial^3\psi\partial^2\psi\rangle \\
&+ 0.388562\,|\partial^4\phi\partial\phi\partial^3\psi\partial\psi\rangle + 0.475889\,|\partial^3\phi\partial^2\phi\partial^3\psi\partial\psi\rangle \\
&- 0.175534\,|\partial^5\phi\partial\phi\partial^2\psi\partial\psi\rangle - 0.222035\,|\partial^4\phi\partial^2\phi\partial^2\psi\partial\psi\rangle - 0.166527\,|\partial^3\phi\partial^3\phi\partial^2\psi\partial\psi\rangle\ .
\end{aligned}
$$

(We have left off the RQ subscripts for conciseness in the above states).

2.4 `MatrixElements-Scalar.wl`

The package `MatrixElements-Scalar.wl` contains routines relevant for computing the ϕ^2 and ϕ^4 matrix elements.

This package contains three main functions that the user can call:

- `ComputeScalarMassMatrix[`Δ`max, basisBoson]`, which computes the mass term matrix elements from the ϕ^2 term. The output is stored in the variable `fullMassMatrix`.
- `ComputePhi4NtoNMatrix[`Δ`max, basisBoson]`, which computes the n-to-n matrix elements from the ϕ^4 interaction term. The output is stored in the variable `fullPhi4NtoNMatrix`.
- `ComputePhi4NtoNPlus2Matrix[`Δ`max, basisBoson]`, which computes the n-to-$(n+2)$ matrix elements from the ϕ^4 interaction term. he output is stored in the variable `fullPhi4NtoNPlus2Matrix`.

The arguments for all of these functions work in the same way: the first argument `Δmax` instructs the desired function to compute the matrix elements up to and including the scaling dimension Δ_{max}. The second argument `basisBoson` is the output of the package `Basis-Scalar.wl` (that is, the scalar basis tabulated up to and including Δ_{max}). Note that this means that the first argument `Δmax` *cannot exceed* the truncation Δ_{max} that was chosen for `basisBoson`!

For example, suppose the user has run `Basis-Scalar.wl` successfully and has obtained the scalar basis up to $\Delta_{\text{max}} = 20$ and it is stored in the variable `basisBoson`. Then, to obtain, e.g. the n-to-n matrix elements, we can input

```
In[9]:= ComputePhi4NtoNMatrix[20,basisBoson]
```

The output is a 627×627 matrix (since there are 627 states at this Δ_{max}) wrapped as a `SparseArray` in the variable `fullPhi4NtoNMatrix`. For example, we can look at the first few elements of this matrix with the command

```
In[10]:= fullPhi4NtoNMatrix[[1;;5,1;;5]] // MatrixForm
```

The conventions for `ComputePhi4NtoNMatrix` and `ComputePhi4NtoNPlus2Matrix` are that all numerical factors except the coupling constant λ are absorbed into the output matrices.

There are several auxiliary functions within `MatrixElements-Scalar.wl` that are used internally within the package. The user may wish to call them, and we list their functionality here for completeness.

- `massNtoN[nR,Δmax]`, which computes the monomial mass matrix. The output is the mass term matrix evaluated for monomials.
- `phi4NtoN[nR,Δmax]`, which computes the monomial n-to-n interaction matrices. The output is the n-to-n matrix evaluated for monomials.
- `phi4NtoNPlus2[nR,Δmax]`, which computes the monomial n-to-$(n+2)$ interaction matrices. Note that the output here is not a square matrix, as there are typically more $(n+2)$-particle monomials than n-particle monomials.
- `phiAnnihilateMap[nR,Δmax]`, which determines the matrix representation of radial quantization annihilation modes (see section 7.6 of main paper). This functions output a sparse array that is the representation of a_k on a set of monomials labelled by occupation number. The related functions `phiAnnihilateTwoMap`, and `phiAnnihilateThreeMap` do the same but with a_k^2 and a_k^3.

The arguments for all of these functions are the same: they compute the desired function a given particle number `nR` and up to $\Delta_{\max}$ `Δmax`. The organization of the monomials for the monomial matrix elements matches that of the basis states.

2.5 `MatrixElements-Mixon-XXX.wl`

There are six packages for mixon matrix elements that all work in essentially the same way, but produce matrix elements for different operators. These are `MatrixElements-Mixon-XXX.wl` with `XXX` = `Mass`,`Phi3`, `Phi4`, `Yukawa`, `QPlus`, and `QMinus` (the "`Mixon`" has been dropped from the package title for the last three of these since there is no scalar-only or fermion-only version of them).

We will describe `MatrixElements-Mixon-Mass.wl` first, and then briefly summarize how the others differ.

`MatrixElements-Mixon-Mass.wl`

The package `MatrixElements-Mixon-Mass.wl` contains routines relevant for computing matrix elements for both the scalar and fermion mass term, where the external states are so-called "mixon" states containing both scalars and fermions. This package contains one main function that the user can call:

- `computeMixonMassMatrices[ΔmaxSuffix,basisMixon,fdr]`, which computes all matrix elements for both the scalar and fermion mass term. The output is two matrices, whose entries correspond to matrix elements between the states in `basisMixon`, which are stored in the variables `matScalarMass` and `matFermionMass`.

The package creates two files to save the output, located in the directory specified by `fdr`, which are labeled by the first argument `ΔmaxSuffix`. For example, for `ΔmaxSuffix`= 20, the package would create the two files:

- `MatrixScalarMassMixonD20`, containing the ϕ^2 matrix elements in the variable `matScalarMass`,
- `MatrixFermionMassMixonD20`, containing the $\psi\frac{1}{\partial}\psi$ matrix elements in the variable `matFermionMass`.

The argument `ΔmaxSuffix` only specifies the names for the output files, and does not affect any part of the evaluation. The package simply computes all matrix elements for the input basis `basisMixon`. Only matrix elements between states with the same number of scalars and fermions will be nonzero.

`MatrixElements-Mixon-Phi3.wl`

The package `MatrixElements-Mixon-Phi3.wl` computes mixon matrix elements for the ϕ^3 interaction. The main user function is:

- `computePhi3Matrices[`Δ`maxSuffix,basisMixon,fdr]`, which computes all matrix elements for the ϕ^3 interaction. The output is a matrix, whose entries correspond to matrix elements between the states in `basisMixon`, which are stored in the variable `matPhi3NtoN1`.

For example, for Δ`maxSuffix`$= 20$, the package would create the file:

- `MatrixPhi3NtoN1MixonD20`, containing the ϕ^3 matrix elements in the variable `matPhi3NtoN1`.

Only matrix elements between states with the same number of fermions and whose number of scalars differ by 1 will be nonzero.

`MatrixElements-Mixon-Phi4.wl`

The package `MatrixElements-Mixon-Phi4.wl` computes mixon matrix elements for the ϕ^4 interaction. The main user function is

- `computePhi4Matrices[`Δ`maxSuffix,basisMixon,fdr]`, which computes all matrix elements for the ϕ^4 interaction. The output is two matrices, whose entries correspond to the $n \to n$ and $n \to n+2$ matrix elements between the states in `basisMixon`, which are stored in the variables `matPhi4NtoN` and `matPhi4NtoN2`.

For example, for Δ`maxSuffix`$= 20$, the package would create the two files:

- `MatrixPhi4NtoNMixonD20`, containing the $n \to n$ matrix elements in the variable `matPhi4NtoN`,
- `MatrixPhi4NtoN2MixonD20`, containing the $n \to n+2$ matrix elements in the variable `matPhi4NtoN2`.

Only matrix elements between states with the same number of fermions and whose number of scalars differ by 0 or 2 will be nonzero.

`MatrixElements-Yukawa.wl`

The package `MatrixElements-Yukawa.wl` computes mixon matrix elements for the Yukawa $\phi\psi\chi$ interaction, which in lightcone becomes two interactions, a cubic $\sim \phi\psi\frac{1}{\partial}\psi$ and a quartic $\sim \phi\psi\frac{1}{\partial}\phi\psi$. The main user function is

- `computeYukawaMatrices[ΔmaxSuffix,basisMixon,fdr]`, which computes all matrix elements for both the cubic and quartic interactions. The output is two matrices, stored in the variables `matYukawaCubic` and `matYukawaQuartic`, for the cubic and quartic matrix elements, respectively.

`MatrixElements-QPlus.wl`

The package `MatrixElements-QPlus.wl` computes mixon matrix elements for the supercharge Q_+ for a mass term $Q_+ \sim \phi\psi$ and a cubic term $Q_+ \sim \phi^2\psi$. The main user function is

- `computeQPlusMatrix[ΔmaxSuffix,basisMixon,fdr]`, which computes all matrix elements for both the mass and cubic terms. The output is two matrices, stored in the variables `matQPlusMass` and `matQPlusInt`.

`MatrixElements-QMinus.wl`

The package `MatrixElements-QMinus.wl` computes mixon matrix elements for the supercharge $Q_- \sim \partial\phi\psi$. The main user function is

- `computeQMinusMatrix[ΔmaxSuffix,basisMixon,fdr]`, which computes all matrix elements for Q_-. The output is a matrix, stored in the variables `matQMinus`.

3 Demos

3.1 Simple Scalar Code

The file `SimpleScalarCode.nb` provides a demonstration of the application of LCT to ϕ^4 theory,

$$\mathcal{L} = \frac{1}{2}\partial_\mu\phi\partial^\mu\phi - \frac{1}{2}m^2\phi^2 - \frac{1}{4!}\lambda\phi^4. \tag{3.1}$$

This notebook is self-contained (i.e. needs no external packages) and uses the example code presented in section 4. The goal of this notebook is to provide a simple, readable example of the main steps in LCT. This code is much less efficient than the main packages, and can only be used to study low values of $\Delta_{\max}$ ($\lesssim 10$).

There are three sections in the notebook:

- **Construct Basis**
  The first section constructs a complete, orthonormal basis of primary states in free scalar field theory, following the methods in sections 4.1 and 4.2. There are

two main functions in this section: `PrimarySetSimp[n,deg]`, which constructs all primary operators with particle number n and scaling dimension $\Delta =$`deg+n`, and `orthoPrimaries[n,deg]` which orthonormalizes the primary operators at level (`n,deg`) with respect to the momentum space inner product. Using these functions, we reproduce the $\Delta_{\max} = 5$ basis in Table 5.

- **Matrix Elements**
  The second section constructs the Hamiltonian matrix elements in the basis of primary operators for the ϕ^2 and ϕ^4 deformations, following the methods of section 4.3 and 4.4. There are three main functions, which all have very similar structure: `PrimaryMassMatrix[n,deg1,deg2]` which computes the ϕ^2 matrix elements between all states at level (`n,deg1`) and those at level (`n,deg2`), `PrimaryNtoNMatrix[n,deg1,deg2]` which computes the ϕ^4 matrix elements between all states at levels (`n,deg1`) and (`n,deg2`), and `PrimaryNtoN2Matrix[n,deg1,deg2]` which computes the ϕ^4 matrix elements between all states at level (`n,deg1`) and those at level (`n+2,deg2`). Using these functions, we reproduce the $\Delta_{\max} = 5$ matrix elements in Tables 7 and 9, and compute the mass gap as a function of the dimensionless ϕ^4 coupling $\frac{\lambda}{4\pi m^2}$.

- **Spectral Densities**
  The final section constructs the overlaps between the operators ϕ^n and the basis of primary operators, following the method of section 4.6. There is one main function: `primaryPhiN[n,deg]` which computes the overlap of ϕ^n with all basis states at level (`n,deg`). Using this function, we compute the integrated spectral density for ϕ^2 in the free massive theory ($\lambda = 0$) with $\Delta_{\max} = 20$.

3.2 2D QCD

In the folder `./documentation/QCD Demo/`, we have included files to work through the 2d massless QCD example presented in section 6 of the main paper. In particular, the notebook `runQCD.nb` includes functions to study the theory described by the Lagrangian

$$\mathcal{L} = i\overline{\Psi}\not{D}\Psi - \frac{1}{2}\mathrm{Tr}F_{\mu\nu}F^{\mu\nu}, \tag{3.2}$$

and produce all figures shown in section 6. We briefly describe the general structure and functions of this notebook below (more comments may be found within the notebooks).

Roughly, `runQCD.nb` is divided into two parts. The first part (**Section 1. The symbolic package**) includes symbolic functions that can be used to compute basis states and matrix elements as symbolic functions of the number of colors N_c. We also

include a demonstration of how to use the functions from the first part. Since everything is done analytically in the first part, there is a trade-off in computation speed. The second part (**Section 2. The numerical package**) of the package includes faster numerical based functions. This requires setting N_c to a desired value before running the functions. Note that none of the functions in `runQCD.nb` use the radial quantization improvements developed in part II of the main paper, which we leave to future work.

In the first part of `runQCD.nb`, we include the following sections:

- **Generate the basis states recursively:** Here we include functions which symbolically compute 2d QCD primaries. The key function `stateSet[dim]` computes primary states at dimension `dim` by taking the double-trace combination of lower dimension primaries, starting with the seed primary $\psi^\dagger\psi$.

- **Inner products, color indices and contractions of spectators:** This section includes functions for computing finite N_c inner products, which are needed to orthogonalize primaries obtained in the previous section.

- **Gauge interaction: structure:** This section includes functions for computing the index structure of monomial gauge interaction matrix elements. It is roughly subdivided into three parts: the first part, which determines the gauge interaction structure for the 'active' part of the matrix element. The second, which determines the the contribution from the 'spectators'. And lastly, we put together these two steps to obtain the structure of final matrix element. More details about the gauge interaction may be found in Appendix D of the main paper.

- **Gauge interaction: active part:** Defines formulas for the active part of the gauge interaction for generic monomials.

- **Primary matrix elements:** Computes primaries, orthgoonalizes them, and puts together gauge interaction monomial matrix elements into gauge interaction matrix elements for primaries.

In the following section (**Use the code**), we include a demonstration of how to use the functions described above, and the output they generate.

In the second part, we import the package `QCD-public.wl`, which essentially includes numerical implementations of the functions in the first part. It includes the following sections:

- **Compute the spectral density:** which computes the stress tensor spectral density shown in Fig. 9 of the main text for $N_c = 3$.

- **Use the spectral density to extract the low energy single particle spectrum:** where we identify single particle states in the stress tensor spectral density for $N_c = 3$ and $N_c = 6$. We produce Fig. 10 of the main text.

Finally, in the standalone notebook `LargeNDemo.nb`, we include expressions to compute analytic matrix elements at large N_c, in both the cosine and LCT basis. Using these matrix elements, we reproduce Figure 8 in [].

3.3 Application I: ϕ^4 Theory

The folder `Phi4Demo/` contains all files necessary to work through the first application presented in Part III, ϕ^4 theory. In particular, the notebook `Phi4Demo.nb` demonstrates how to use the packages `Basis-Scalar.wl` and `MatrixElements-Scalar.wl` to reproduce all figures in section 9.

The notebook `Phi4Demo.nb` is divided into two main parts:

- **Generate Basis and Matrix Elements**
  The first part uses `Basis-Scalar.wl` to construct the complete basis of states in free scalar theory up to the scaling dimension `DMAX` (set by the user at the top of the notebook), then uses `MatrixElements-Scalar.wl` to compute all ϕ^2 and ϕ^4 matrix elements for this basis. The output basis and matrix elements are saved in files labeled by `DMAX`. For example, with `DMAX`= 20 (the default value) we obtain the files:

  - `BasisBosonD20.WXF`, containing the basis states,
  - `ScalarMassD20.WXF`, containing the ϕ^2 matrix elements,
  - `ScalarPhi4NtoND20.WXF`, containing the $n \to n$ ϕ^4 matrix elements,
  - `ScalarPhi4NtoN2D20.WXF`, containing the $n \to n + 2$ ϕ^4 matrix elements.

  The default value `DMAX`= 20 allows the user to keep the runtime and file sizes relatively small. To completely reproduce the figures in section 9, the user should set `DMAX`= 40.

- **Diagonalize Hamiltonian and Reproduce Figures**
  The second part uses the output data from the first part to reproduce Figures 12-17. For each figure, the user can study results for any value of $\Delta_{\text{max}} \leq$`DMAX`. Within this part, there are three main sections:

  - **Mass Gap vs Coupling (Figure 12)**
    First, we diagonalize the full Hamiltonian for a range of values for the coupling λ. Because the theory has a $\mathbb{Z}_2$ symmetry $\phi \to -\phi$, the Hamiltonian

can be split into odd and even particle number sectors, which can be diagonalized independently. We use the results to reproduce Figure 12, showing the one-, two-, and three-particle thresholds as a function of λ.

- **Convergence with $\Delta_{\max}$ (Figure 13)**
  Next, we diagonalize the Hamiltonian at a fixed value of λ (set by the user) for various values of $\Delta_{\max}$. We extrapolate these results to $\Delta_{\max} \to \infty$ by fitting the error as $1/\Delta_{\max}^p$, where the user can vary p. We use these results to reproduce Figure 13, showing the convergence of the one-, two-, and three-particle thresholds with increasing $\Delta_{\max}$.
- **Spectral Densities (Figures 14-17)**
  Finally, we diagonalize the Hamiltonian at a fixed value of λ and compute spectral densities of the stress-energy tensor. We compute the Zamolodchikov C-function (Figures 14-16) as well as the integrated spectral density of the trace T_{+-} (Figure 17). In comparing spectral density results for different values of $\Delta_{\max}$, it is best to keep the mass gap μ^2_{gap} fixed, rather than the coupling λ. In order to do this, we also provide a section at the end which allows the user to specify $\Delta_{\max}$ and a desired value for μ^2_{gap} (in units of the bare mass). The code then scans over λ to construct an interpolating function $\lambda(\mu^2_{\text{gap}})$ and determines the value of the coupling needed to obtain the desired mass gap.

3.4 Application II: Yukawa Theory

There are two mathematica notebooks that demonstrate different parts of the non-SUSY Yukawa example described by the Lagrangian

$$\mathcal{L} = \frac{1}{2}(\partial\phi)^2 - \frac{1}{2}m_\phi^2\phi^2 + \sqrt{2}i\psi\partial_+\psi - \frac{1}{\sqrt{2}}\psi\frac{m_\psi^2}{i\partial}\psi - \sqrt{2}m_\psi g\phi\psi\frac{1}{i\partial}\psi - \frac{g^2}{\sqrt{2}}\phi\psi\frac{1}{i\partial}\phi\psi. \quad (3.3)$$

`YukawaOneLoopMassShifts.nb` The file is in the path `./documentation/YukawaOneLoopMassShifts.nb`. The file walks through the calculation of second order time-independent perturbation theory to compute the one-loop mass shift of the one-fermion state, and reproduces Fig. 19 of the main text. The notebook does not use any packages.

`Yukawa_Demo.nb` The file is in the path `./documentation/Yukawa Demo/Yukawa_Demo`. This is the main file that walks through the computation of LCT results of the Yukawa theory using the code packages.

The file contains the following parts:

- **How to run the Yukawa package**
  This section introduces the basics of running the Yukawa packages. The reader can directly run the whole section and see instant results.

  - **Run packages**
    This subsection introduces the basic commands to run the packages to compute the basis states and matrix elements at certain $\Delta_{\max}$. The default set is $\Delta_{\max} = 6$ (set at the beginning) so running the whole section is instant. If the reader wishes to generate the data to reproduce the results in the main text, one can change this line to $\Delta_{\max} = 20$ and it takes about 3 hours to finish.

  - **Parse the computed data into readable forms**
    The raw output data of `computeXXXX[...]` is hard to use. This subsection introduces the useful commands to flatten the states list and interaction matrix into readable forms which are used throughout the rest of the notebook.

  - **Save data in more compact format**
    This subsection introduces the convention of importing/exporting data computed in previous subsections. Running this subsection will create (by default) $\Delta_{\max} = 6$ data in exactly the same form as will be imported in the next section. Setting $\Delta_{\max} = 20$ will override the pre-generated data.

- **Load previously computed data and parse**
  This section loads the pre-generated (or user-generated if one sets $\Delta_{\max} = 20$) data at $\Delta_{\max} = 20$. Readers need to run this section before proceeding to later sections.

- **Fermion Mass shift (Figures 20-21)**
  This section computes the fermion mass shift for fermion multi-particle states, using the second order time-independent perturbation theory and the explicit Hamiltonian at $\Delta_{\max}$.

  - **Old fashioned perturbation theory**
    This subsection defines and computes the intermediate results used in perturbation theory.

  - **Fig 20**
    This subsection tests two different schemes to cancel the fermion mass one-loop divergence – using a local counter-term or a non-local one, by computing the one-loop fermion mass shift of 1- 2- and 3- particle states.

- **Fig 21**
  This subsection computes the one-loop fermion mass shift of 1- 2- and 3-particle states in the above two schemes. In addition to the divergence-canceling counter-terms, a finite local term is introduced to restore Lorentz covariance.

- **The energy spectrum (Figure 22)**
  This section computes the low energy spectrum of Yukawa theory at $m_\psi/m_\phi = 0.4$ and 0.8, as a function of the coupling, g. This section may take a few minutes to run. In order to speed up, the scanning of g takes larger step length than the actual plot in the main text.

- **Spectral density (Figures 23-24)**
  This section computes the spectral density.

  - **Fig 23 - C function**
    This subsection computes the integrated spectral density of T_{--}, known as the Zamolodchikov C function. The plots shows the data at $\Delta_{\max} = 20$ and $m_\psi/m_\phi = 0.4$ and 0.8.
  - **Fig 24 - $\langle\phi\phi\rangle$ correlator**
    This subsection computes the spectral density of ϕ operator. The plots shows the data at $\Delta_{\max} = 20$ and $m_\psi/m_\phi = 0.4$ at different g. The spectral density is compared with the Breit-Wigner distribution.

4 Troubleshooting

We can add more when our beta testers get back to us. Here we enumerate potential errors / commonly asked questions that can arise when trying to use LCT code:

- The function `BinarySerialize` or `BinaryDeserialize` gives a "Serialized data is corrupt and does not represent an expression" error.

  This is likely due to running the code on an earlier version of Mathematica. LCT packages have been tested on Mathematica version 12.1.0, and older versions may generate error messages with `BinarySerialize` or `BinaryDeserialize` functions, especially when changing precision. We recommend updating Mathematica to 12.0.0 or later.

- Functions do not seem to compile / external libraries necessary for `CompilationTarget` cannot be found.

  In order to optimize certain operations, LCT packages pre-compile frequently called Mathematica functions using the `Compile` function. These functions are translated into C code with the command `CompilationTarget -> "C"`. This assumes that the user's machine has a suitable external C compiler. If one is not found, Mathematica defaults to `CompilationTarget -> "WVM"` which creates code for the traditional Wolfram System virtual machine, at the expense of computational efficiency. We recommend installing a C compiler for fastest results.