Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2016-0107

Vol. **22**, No. 1, pp. 1-38
July 2017

PyCFTBoot: A Flexible Interface for the Conformal Bootstrap

Connor Behan*

Department of Physics and Astronomy, Stony Brook University, Stony Brook, NY 11790, USA.

Received 15 July 2016; Accepted (in revised version) 7 November 2016

Abstract. We introduce PyCFTBoot, a wrapper designed to reduce the barrier to entry in conformal bootstrap calculations that require semidefinite programming. Symengine and SDPB are used for the most intensive symbolic and numerical steps respectively. After reviewing the built-in algorithms for conformal blocks, we explain how to use the code through a number of examples that verify past results. As an application, we show that the multi-correlator bootstrap still appears to single out the Wilson-Fisher fixed points as special theories in dimensions between 3 and 4 despite the recent proof that they violate unitarity.

AMS subject classifications: 65Z05, 68W30, 81T80, 90C34

Key words: CFT, conformal bootstrap, python, semidefinite programming.

1 Introduction

The conformal bootstrap [1, 2] has joined holography [3] as one of the most important tools for understanding strongly coupled conformal field theories (CFTs) in higher dimensions. Much of the progress comes from a numerical procedure initiated in [4], which exploits the constraints of crossing symmetry and unitarity. This has been successfully used to bound scaling dimensions and three point function coefficients in a wide range of conformal [5–16] and superconformal [17–21] theories in dimensions between 2 and 6. The first widely released code designed to perform these calculations was `JuliBoots` [22], a conformal bootstrap package based around a linear program solver. Shortly afterward, the solver `SDPB` [23] was released, giving the community access to the semidefinite programming methods pioneered in [10, 18, 24]†.

*Corresponding author. *Email address:* `connor.behan@gmail.com` (C. Behan)

†Readers interested in conformal blocks for their role in algebraic geometry might appreciate the [25] package.

 ©2017 Global-Science Press

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

The advantages of the two are largely complementary. Semidefinite programming has superior performance in systems with multiple crossing equations and it is currently the only technique which extracts information from correlators of operators with different scaling dimensions [24]. As such, `SDPB` has become the standard code for most numerical bootstrap studies in the last year [26–34]. Unlike `JuliBoots` however, it does not provide simple methods for specifying important kinematics information. Included in this are the crossing equations which depend on the type of CFT being studied and conformal blocks, special functions that depend on the dimension of space and a number of accuracy parameters. All of the above studies have performed these calculations using customized scripts for `Mathematica`. A new program, aiming to reduce this duplication of effort, is `PyCFTBoot` written in Python. Realizing a hope of [22], it handles the computer algebra that goes into a numerical bootstrap entirely with free software. `PyCFTBoot` may be downloaded from

`https://github.com/cbehan/pycftboot`,

where all future development is expected to take place. Besides `SDPB`, a few other dependencies are required in order to use it.

In mathematical Python software, `numpy` [35] and `sympy` [36] are two widely used packages that come to mind. Both of them are needed by `PyCFTBoot`. However, `sympy` is not fast enough to generate large tables of conformal blocks. It is only used in a few non-critical places that need to call Gegenbauer polynomials or the incomplete gamma function. Instead, the bulk of the symbolic algebra is handled by a fast C++ library called `symengine`. Python bindings have been chosen (over Ruby and Julia) because they are the most mature at the time of writing. These less common packages are downloadable from

`https://github.com/symengine/symengine` (last tested: 5427bbe),
`https://github.com/symengine/symengine.py` (last tested: 9d23ef7).

Surprises are most easily avoided by using `PyCFTBoot` with Python 2.7 on GNU / Linux, but it has also been tested with Python 3.5. Descriptions of the important functions, included in the source code, may be viewed with the Python documentation server. Additionally, readers who are anxious to try the bootstrap may follow the commented tutorial distributed alongside the main file.

In Section 2 of this note, we describe the algorithms that have been chosen to generate derivatives of conformal blocks and report some rough performance figures. Section 3 explains how semidefinite programs are formulated from these tables. In describing the main SDP object, it contains a few parts that read like passages from a user manual. Some examples, worked out in Section 4, demonstrate that most of the known bootstrap results to date can in principle be reproduced with `PyCFTBoot`. Before we conclude, Section 5 extends a previous result in the literature by using `PyCFTBoot` to probe the "islands" of allowed critical exponents in dimensions between 3 and 4 [37].

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

2 Conformal blocks

Unlike with two or three point functions, conformal kinematics only determine the four point function up to an arbitrary dependence on two variables. Specifically for scalars[‡],

$$\langle\phi_1(x_1)\phi_2(x_2)\phi_3(x_3)\phi_4(x_4)\rangle = \left(\frac{|x_{24}|}{|x_{14}|}\right)^{\Delta_{12}}\left(\frac{|x_{14}|}{|x_{13}|}\right)^{\Delta_{34}}\frac{g(u,v)}{|x_{12}|^{\Delta_1+\Delta_2}|x_{34}|^{\Delta_3+\Delta_4}}, \tag{2.1}$$

where $u=x_{12}^2x_{34}^2/x_{13}^2x_{24}^2$ and $v=x_{14}^2x_{23}^2/x_{13}^2x_{24}^2$. As explained in the seminal works [50,51] on (global) conformal blocks, $g(u,v)$ may be expanded in a convergent series with each term coming from a primary operator in the theory. This is done by way of the operator product expansion (OPE):

$$\phi_1(x)\phi_2(0) = \sum_{\mathcal{O}}\frac{\lambda_{12\mathcal{O}}}{|x|^{\Delta_1+\Delta_2-\Delta}}C_{\mathcal{O}}^{\mu_1\cdots\mu_\ell}(x,\partial)\mathcal{O}_{\mu_1\cdots\mu_\ell}(0). \tag{2.2}$$

Using this in the $(12)(34)$ channels for example produces

$$g(u,v) = \sum_{\mathcal{O}}\lambda_{12\mathcal{O}}\lambda_{34\mathcal{O}}g_{\mathcal{O}}^{\Delta_{12},\Delta_{34}}(u,v),$$

where each function depends on the spatial dimension d or equivalently on $\nu=\frac{d-2}{2}$. The subscript $\mathcal{O}$ is often written as (Δ,ℓ) since all primary operators that couple to scalars transform in some spin-ℓ representation of $SO(d)$. Crossing symmetry is the statement that all three choices for the OPE channels must agree. This is what leads to the bootstrap but we will postpone a discussion of this to the next section.

2.1 Rational approximations

Rather than the cross-ratios u and v, conformal blocks are most often considered as functions of z and $\bar{z}$, defined by using conformal transformations to send x_1, x_3 and x_4 to 0, 1 and ∞ respectively. The blocks are analytic for $0<z,\bar{z}<1$ and most bootstrap studies focus on the crossing symmetric point $(z_*,\bar{z}_*)=(\frac{1}{2},\frac{1}{2})$. Although there are other useful variables [52], Table 1 shows all of the co-ordinates used by `PyCFTBoot`.

As observed in [10, 53], a block may be expanded in powers of r where each term corresponds to a new descendant in the multiplet of $\mathcal{O}$. As the scaling dimension Δ is varied, coefficients in the sum diverge at certain non-unitary values. When they do, the residue is proportional to a conformal block itself. This motivated [10] to develop the recurrence relations

$$h_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(r,\eta) \equiv r^{-\Delta}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(r,\eta), \tag{2.3a}$$

$$h_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(r,\eta) = h_{\infty,\ell}^{\Delta_{12},\Delta_{34}}(r,\eta) + \sum_i\frac{c_i^{\Delta_{12},\Delta_{34}}(\ell)r^{n_i}}{\Delta-\Delta_i(\ell)}h_{\Delta_i(\ell)+n_i,\ell_i}^{\Delta_{12},\Delta_{34}}(r,\eta). \tag{2.3b}$$

[‡]We focus on the scalar correlators currently supported by `PyCFTBoot` but it would be very interesting to incorporate the ongoing work regarding operators with spin [29,38–49].

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

Table 1: Useful variables for four point conformal blocks in terms of z and $\bar{z}$.

First	Second	Crossing point
$u=\lvert z\rvert^2$	$v=\lvert 1-z\rvert^2$	$(u_*,v_*)=\left(\frac{1}{4},\frac{1}{4}\right)$
$a=z+\bar{z}$	$b=(z-\bar{z})^2$	$(a_*,b_*)=(1,0)$
$\rho=\frac{z}{(1+\sqrt{1-z})^2}$	$\bar{\rho}=\frac{\bar{z}}{(1+\sqrt{1-\bar{z}})^2}$	$(\rho_*,\bar{\rho}_*)=(3-2\sqrt{2},3-2\sqrt{2})$
$r=\lvert\rho\rvert$	$\eta=\frac{\rho+\bar{\rho}}{2\lvert\rho\rvert}$	$(r_*,\eta_*)=(3-2\sqrt{2},1)$

Table 2: The three types of poles in Δ for the meromorphic conformal blocks. Two of them have infinitely many elements labelled by the integer $k>0$. The third type requires $0<k\leq\ell$.

n_i	$\Delta_i(\ell)$	ℓ_i	$c_i^{\Delta_{12},\Delta_{34}}(\ell)$
k	$1-\ell-k$	$\ell+k$	$c_1^{\Delta_{12},\Delta_{34}}(\ell,k)$
$2k$	$1+\nu-k$	ℓ	$c_2^{\Delta_{12},\Delta_{34}}(\ell,k)$
k	$1+\ell+2\nu-k$	$\ell-k$	$c_3^{\Delta_{12},\Delta_{34}}(\ell,k)$

The leading term is given by [24]

$$h_{\infty,\ell}^{\Delta_{12},\Delta_{34}}(r,\eta)=\frac{\ell!}{(2\nu)_\ell}\frac{(-1)^\ell C_\ell^\nu(\eta)}{(1-r^2)^\nu(1+r^2+2r\eta)^{\frac{1}{2}(1+\Delta_{12}-\Delta_{34})}(1+r^2-2r\eta)^{\frac{1}{2}(1-\Delta_{12}+\Delta_{34})}}. \tag{2.4}$$

Table 2 describes the data needed to construct the poles and residues in (2.3). These were noticed empirically in [24] but most of them were later proven in [54]. We must use

$$c_1^{\Delta_{12},\Delta_{34}}(\ell,k)=-\frac{k(-4)^k}{(k!)^2}\frac{(\ell+2\nu)_k}{(\ell+\nu)_k}\left(\frac{1}{2}(1-k+\Delta_{12})\right)_k\left(\frac{1}{2}(1-k+\Delta_{34})\right)_k, \tag{2.5a}$$

$$\begin{aligned}c_2^{\Delta_{12},\Delta_{34}}(\ell,k)=&\frac{k(\nu+1)_{k-1}(-\nu)_{k+1}}{(k!)^2}\frac{\ell+\nu-k}{\ell+\nu+k}\left(\frac{\ell+\nu-k+1}{2}\right)_k^{-2}\left(\frac{\ell+\nu-k}{2}\right)_k^{-2}\\&\times\left(\frac{1}{2}(1-k+\ell-\Delta_{12}+\nu)\right)_k\left(\frac{1}{2}(1-k+\ell+\Delta_{12}+\nu)\right)_k\\&\times\left(\frac{1}{2}(1-k+\ell-\Delta_{34}+\nu)\right)_k\left(\frac{1}{2}(1-k+\ell+\Delta_{34}+\nu)\right)_k,\end{aligned} \tag{2.5b}$$

$$c_3^{\Delta_{12},\Delta_{34}}(\ell,k)=-\frac{k(-4)^k}{(k!)^2}\frac{(\ell+1-k)_k}{(\ell+\nu+1-k)_k}\left(\frac{1}{2}(1-k+\Delta_{12})\right)_k\left(\frac{1}{2}(1-k+\Delta_{34})\right)_k \tag{2.5c}$$

to fill in the last column. One fact that can be seen from (2.5) is that $c_1^{0,0}(\ell,k)$ and $c_3^{0,0}(\ell,k)$ are only non-zero when k is even. This means that when the external scalars are identical, blocks of even and odd spin do not show up in each other's recurrence relations. Consequently, adjusting the overall normalization of $h_{\Delta,\ell}^{0,0}(r,\eta)$ by $(-1)^\ell$ is equivalent to simply removing the factor of $(-1)^\ell$ from (2.4). Indeed, for many studies involving identical

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

scalars, it was not present. The generalization to non-zero dimension differences shows us that more drastic changes would be needed if we still wanted to cancel the $(-1)^\ell$ in (2.4). Therefore `PyCFTBoot` keeps it around. The end of this paper points out the examples in which this subtlety needs to be remembered.

For spins up to some $\ell_{\max}$, we need to know several derivatives of $h^{\Delta_{12},\Delta_{34}}_{\Delta,\ell}$ evaluated at $(r_*,\eta_*)=(3-2\sqrt{2},1)$. If we evaluated (2.3) for powers of r up to $k_{\max}$ and differentiated after, we would suffer a large performance hit. This is because there would be many appearances of (2.4)'s non-polynomial contributions all multiplied by different powers of r. A better strategy is to compute all derivatives at the same time via matrix multiplication [11]. To this end, we define the vector $\mathbf{h}_{\infty,\ell}$ with all desired derivatives of (2.4) already evaluated at the crossing point. They are grouped into "chunks" of ∂_r powers for a given number of ∂_η powers[§]. For example, a computation going up to third order would set

$$\mathbf{h}_{\infty,\ell}=\left[1,\frac{\partial}{\partial r},\frac{\partial^2}{\partial r^2},\frac{\partial^3}{\partial r^3},\frac{\partial}{\partial \eta},\frac{\partial^2}{\partial \eta\partial r},\frac{\partial^3}{\partial \eta\partial r^2},\frac{\partial^2}{\partial \eta^2},\frac{\partial^3}{\partial \eta^2\partial r},\frac{\partial^3}{\partial \eta^3}\right]^{\mathrm{T}} h_{\infty,\ell}. \tag{2.6}$$

Seeing what happens when we differentiate $r^{n_i}h_{\Delta,\ell}$ several times, the matrix telling us what linear combination of derivatives to take is

$$\mathbf{R}^{n_i}=\begin{bmatrix} r_*^{n_i} & 0 & 0 & \cdots \\ n_i r_*^{n_i-1} & r_*^{n_i} & 0 & \cdots \\ n_i(n_i-1)r_*^{n_i-2} & 2n_i r_*^{n_i-1} & r_*^{n_i} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}=\begin{bmatrix} r_* & 0 & 0 & \cdots \\ 1 & r_* & 0 & \cdots \\ 0 & 2 & r_* & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}^{n_i}. \tag{2.7}$$

This is the matrix acting on a single chunk. Since η is unaffected, the full $\mathbf{R}$ is the tensor product of (2.7) with the identity. There is a problem with simply writing

$$\mathbf{h}_{\Delta,\ell}=\mathbf{h}_{\infty,\ell}+\sum_i \frac{c_i(\ell)\mathbf{R}^{n_i}}{\Delta-\Delta_i(\ell)}\mathbf{h}_{\Delta_i(\ell)+n_i,\ell_i} \tag{2.8}$$

and repeating this calculation every time a new block appears. It is most easily seen if we compare the number of matrix multiplications involved to the number of unique $\mathbf{h}_{\Delta_i+n_i,\ell_i}$ terms introduced by the recursion. Looking at (2.5), we see a residue $c_2(\ell,k)$ which may vanish sometimes and a residue $c_3(\ell,k)$ which only exists for certain spins. Therefore, the best case scenario (only using $c_1(\ell,k)$) tells us that the number of matrix multiplications # satisfies

$$\#(0)=1, \tag{2.9a}$$

$$\#(k_{\max})>\sum_{k=0}^{k_{\max}-1}\#(k). \tag{2.9b}$$

[§]Although we describe the general case here, we will soon see that normal use of `PyCFTBoot` will only involve one chunk.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

This is the same relation satisfied by the partition function which counts the number of ways to write an integer as the sum of smaller ones. The well known asymptotics of this function [55], tell us that duplicated matrix multiplications will abound by many orders of magnitude with this naive method. Instead `PyCFTBoot` again follows [11] and predicts which residues will be needed ahead of time. This is simply a matter of letting the spin take values $\ell \leq \ell_{\text{max}} + k_{\text{max}}$ for a table whose final entries describe spins up to ℓ_{max}. For each value of ℓ, we let the index i run over all admissible poles in Table 2 and define the residue vectors $\mathbf{d}_{\ell,i}$. All of these are initialized to $\mathbf{h}_{\infty,\ell_i}$. It is then straightforward to iterate

$$\mathbf{d}_{\ell,i} = c_i(\ell)\mathbf{R}^{n_i}\left[\mathbf{h}_{\infty,\ell_i} + \sum_j \frac{\mathbf{d}_{\ell_i,j}}{\Delta_i(\ell)+n_i-\Delta_j(\ell_i)}\right] \tag{2.10}$$

and stop once enough powers of $\mathbf{R}$ are introduced. Rather than updating the residues right away, we consider all $\mathbf{d}_{\ell,i}$ on the right hand side to be the "old values" and replace them with the "new values" once everything on the left hand side has been calculated. These go into the expression

$$\mathbf{h}_{\Delta,\ell} = \mathbf{h}_{\infty,\ell} + \sum_i \frac{\mathbf{d}_{\ell,i}}{\Delta-\Delta_i(\ell)}. \tag{2.11}$$

It is clear that the entries in $\mathbf{h}_{\Delta,\ell}$ are rational functions of Δ. They all have different numerators and the same denominator. Instead of computing (2.11) as written and taking extra time to extract the numerator and denominator, `PyCFTBoot` stores them separately from the start. The leading term of (2.11) is multiplied by $\prod_j(\Delta-\Delta_j(\ell))$ and the i^{th} term of it is multiplied by $\prod_{j\neq i}(\Delta-\Delta_j(\ell))$.

There is a modification to (2.11) that can be used to produce polynomials of smaller degree. Described in [10], it slightly increases the time needed to generate a conformal block table but it can greatly decrease the running time of `SDPB`. The idea is to split the set of poles $\mathcal{P}$ into "large and small" types and use the poles of $\mathcal{P}_>$ to approximate those in $\mathcal{P}_<$. As our criterion, we check whether the maximum component of $\mathbf{d}_{\ell,i}$ is above or below some cutoff θ. For $\Delta_i \in \mathcal{P}_<$, we attempt to choose the $a_{i,k}$ coefficients optimally in

$$\frac{1}{\Delta-\Delta_i} \approx \sum_{\Delta_k\in\mathcal{P}_>} \frac{a_{i,k}}{\Delta-\Delta_k}. \tag{2.12}$$

Following the choice in [10], we demand that the first $|\mathcal{P}_>|/2$ derivatives of (2.12) hold exactly at $\Delta = \Delta_{\text{unitary}} + \theta$ and $\Delta = \theta^{-1}$. If $|\mathcal{P}_>|$ is odd, the last of these derivatives will only hold at one of the points. Once the $a_{i,k}$ are determined by this invertible linear system, `PyCFTBoot` incorporates them into the calculation of (2.11). Whenever it needs to multiply by $\prod_{\Delta_j\neq\Delta_i}(\Delta-\Delta_j)$ and $\Delta_i \in \mathcal{P}_<$, it instead multiplies by $\sum_{\Delta_k\in\mathcal{P}_>} a_{i,k}\prod_{\Delta_j\in\mathcal{P}_>\setminus\{\Delta_k\}}(\Delta-\Delta_j)$.

After the (2.11) computation with the optional degree reduction step, one must obtain a vector $\mathbf{g}_{\Delta,\ell}$ of true conformal block derivatives from its meromorphic version $\mathbf{h}_{\Delta,\ell}$. This is done by restoring the r_*^Δ singularity with another matrix. Specifically,

$$\mathbf{g}_{\Delta,\ell} = r_*^\Delta \mathbf{S}\mathbf{h}_{\Delta,\ell}. \tag{2.13}$$

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

It is easy to see that $r_*^\Delta \mathbf{S}$ must be the same matrix as $\mathbf{R}^{n_i}$ in (2.7) with n_i replaced by Δ. There is no need to build up $\mathbf{S}$ by repeatedly multiplying some simpler matrix by itself. Its (i,j) element is immediately known to be

$$\frac{\Delta\cdots(\Delta-j)}{r_*^j}\binom{i}{j}.$$

Elements of the conformal block vector continue to be rational functions. However, if all numerators in $\mathbf{h}_{\Delta,\ell}$ have the same degree, those in $\mathbf{g}_{\Delta,\ell}$ will have a degree that increases with the order of the derivative. Looking at these numerators, the end result is something of the form

$$\frac{\partial^{m+n}}{\partial\eta^m\partial r^n}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(r_*,\eta_*)=\chi_\ell(\Delta)P_\ell^{\Delta_{12},\Delta_{34};mn}(\Delta), \tag{2.14}$$

which is a polynomial times the positive function

$$\chi_\ell(\Delta)=r_*^\Delta\prod_j(\Delta-\Delta_j(\ell))^{-1}.$$

This is precisely the form required for a task that involves semidefinite programming.

2.2 Further processing

Going from the $(12)(34)$ to the $(14)(23)$ channel switches $u\leftrightarrow v$ and modifies the prefactor in the four point function (2.1). Crossing equations are obtained by setting the differences of these four point functions to zero. The simplest crossing equation with no global symmetry is

$$v^{\frac{\Delta_2+\Delta_3}{2}}g_{1234}(u,v)-u^{\frac{\Delta_1+\Delta_2}{2}}g_{3214}(v,u)=0$$

in [24]. As a result, functions of the form

$$F_{\pm,\Delta,\ell}(u,v)=v^{\Delta_\phi}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(u,v)\pm u^{\Delta_\phi}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(v,u), \tag{2.15}$$

are the natural objects to consider once conformal blocks are known. These have come to be called convolved conformal blocks [22]. In principle, convolved conformal blocks and their derivatives could be calculated directly from the (2.14) result with its r and η variables. However, the simple $u\leftrightarrow v$ transformation is represented by r and η in a much more complicated way. When the second half of (2.15) involves a new function $g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(\tilde{r}(r,\eta),\tilde{\eta}(r,\eta))$, much of the work that goes into the $\frac{\partial^{m+n}}{\partial\eta^m\partial r^n}F_{\pm,\Delta,\ell}(r_*,\eta_*)$ calculation will be spent differentiating $\tilde{r}$ and $\tilde{\eta}$. This extra work during the convolution step can be eliminated if we instead add extra work during the conformal block step to convert (2.14) to $(z,\bar{z})$ or (a,b) variables. At first glance, it might seem that the benefit of this choice is purely organizational — it allows the fast and slow calculations in `PyCFTBoot` to be conceptually separate. As we now discuss however, there is another recurrence relation which gives us a much stronger incentive to change variables.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

Conformal blocks are eigenfunctions of the quadratic Casimir [51]:

$$\left[D_z+D_{\bar{z}}+2\nu\frac{z\bar{z}}{z-\bar{z}}\left((1-z)\frac{\mathrm{d}}{\mathrm{d}z}-(1-\bar{z})\frac{\mathrm{d}}{\mathrm{d}\bar{z}}\right)\right]g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}=c_2 g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}. \tag{2.16}$$

Here, the definitions

$$D_z=(1-z)z^2\frac{\mathrm{d}^2}{\mathrm{d}z^2}+\left(\frac{1}{2}\Delta_{12}-\frac{1}{2}\Delta_{34}-1\right)\frac{\mathrm{d}}{\mathrm{d}z}+\frac{1}{4}\Delta_{12}\Delta_{34}z, \tag{2.17a}$$

$$c_2=\frac{1}{2}\left[\ell(\ell+2\nu)+\Delta(\Delta-2-2\nu)\right], \tag{2.17b}$$

are standard. The existence of a linear differential equation satisfied by the blocks suggests the possibility of building up high order derivatives from lower ones. We may pretend for a minute that $g_{\Delta,\ell}$, $\frac{\partial g_{\Delta,\ell}}{\partial z}$ and $\frac{\partial^2 g_{\Delta,\ell}}{\partial z\partial\bar{z}}$ are all known at $\left(\frac{1}{2},\frac{1}{2}\right)$. The content of (2.16) is then to tell us what $\frac{\partial^2 g_{\Delta,\ell}}{\partial z^2}$ is at the same point. We could attempt to continue this pattern by differentiating (2.16) with respect to z but then $\frac{\partial^3 g_{\Delta,\ell}}{\partial z^3}$ would not be the only unknown derivative anymore. The presence of new unknowns like $\frac{\partial^3 g_{\Delta,\ell}}{\partial\bar{z}^2\partial z}$ forces us to use something more clever.

Such cleverness was found by [56] in which the quadratic and quartic Casimirs of the conformal group are used together. This reveals an ordinary differential equation satisfied by the blocks on the $z=\bar{z}$ diagonal. In terms of the a co-ordinate, this new equation (which clearly keeps new derivatives under control) is

$$D_a^{(4,3)}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}=0, \tag{2.18a}$$

$$D_a^{(4,3)}\equiv\left(\frac{a}{2}-1\right)^3 a^4\frac{\mathrm{d}^4}{\mathrm{d}a^4}+p_3\left(\frac{a}{2}-1\right)^2 a^3\frac{\mathrm{d}^3}{\mathrm{d}a^3}+p_2\left(\frac{a}{2}-1\right)a^2\frac{\mathrm{d}^2}{\mathrm{d}a^2}+p_1 a\frac{\mathrm{d}}{\mathrm{d}a}+p_0. \tag{2.18b}$$

The polynomials $p_0,\cdots,p_3$ used by `PyCFTBoot` are the ones in [56] except with a slight change: they are written with $\frac{a}{2}$ in place of z and multiplied by 8 to force as many coefficients as possible to still be integers. Differentiating (2.18), a fifth derivative of $g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}$ becomes the highest order term. However, the lowest order term continues to be a zeroth derivative. Because $p_0(a)$ has degree 3, our equation only stops having non-derivative terms once it goes up to $\frac{\mathrm{d}^8 g_{\Delta,\ell}}{\mathrm{d}a^8}$. This means that the m^{th} diagonal derivative is calculated from the $\min(m,7)$ lower ones using a handful of simple polynomials. One only needs m to be at least 4 in order to start this process. Because of this, vectors in the slow original recursion (2.10) only need to fit four ∂_r powers. Once the a derivatives are known, more recurrence relations determine the b derivatives. Defining

$$S=-\frac{1}{2}(\Delta_{12}-\Delta_{34})\quad\text{and}\quad P=-\frac{1}{2}\Delta_{12}\Delta_{34},$$

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

we use

$$\begin{aligned}
&2(1-2n-2\nu)\frac{\partial^{m+n}g_{\Delta,\ell}}{\partial a^m\partial b^n}\\
=&2m(1-2n-2\nu)\left[-\frac{\partial^{m+n-1}g_{\Delta,\ell}}{\partial a^{m-1}\partial b^n}+(m-1)\frac{\partial^{m+n-2}g_{\Delta,\ell}}{\partial a^{m-2}\partial b^n}+(m-1)(m-2)\frac{\partial^{m+n-3}g_{\Delta,\ell}}{\partial a^{m-3}\partial b^n}\right]\\
&+\frac{\partial^{m+n+1}g_{\Delta,\ell}}{\partial a^{m+2}\partial b^{n-1}}-(6-m-4n+2\nu+2S)\frac{\partial^{m+n}g_{\Delta,\ell}}{\partial a^{m+1}\partial b^{n-1}}\\
&-\left[4c_2+m^2+8mn-5m+4n^2-2n-2\right.\\
&\left.-4\nu(1-m-n)+4S(m+2n-2)+2P\right]\frac{\partial^{m+n-1}g_{\Delta,\ell}}{\partial a^m\partial b^{n-1}}\\
&-m\left[m^2+12mn-13m+12n^2-34n+22\right.\\
&\left.-2\nu(2n-m-1)+2S(m+4n-5)+2P\right]\frac{\partial^{m+n-2}g_{\Delta,\ell}}{\partial a^{m-1}\partial b^{n-1}}\\
&+(1-n)\left[\frac{\partial^{m+n}g_{\Delta,\ell}}{\partial a^{m+2}\partial b^{n-2}}-(6-3m-4n+2\nu-2S)\frac{\partial^{m+n-1}g_{\Delta,\ell}}{\partial a^{m+1}\partial b^{n-2}}\right].
\end{aligned}\tag{2.19}$$

This is the transverse derivative recursion found in [9] generalized to unequal external dimensions with the different definition of c_2 taken into account. It follows from going back to the original Casimir PDE (2.16) in the (a,b) co-ordinates. The same coefficients can also be found in recent versions of the [22] source code. The form of (2.19) tells us the shape that will be taken by a lattice of derivatives we compute this way. When we make m as high as possible for a given n, the right hand side shows that 2 must be added to reach the highest possible m for $n-1$. This leads to the triangle

$$n\in\{0,\cdots,n_{\max}\},\tag{2.20a}$$
$$m\in\{0,\cdots,2(n-n_{\max})+m_{\max}\},\tag{2.20b}$$

depending on two user-defined parameters. As found in [57], a high $n_{\max}$ is more important than a high $m_{\max}$. An obvious point worth remembering is that (2.18) and (2.19) are only satisfied by exact conformal blocks, not their rational approximations. As a result, these recursions are only valid for computing derivatives if $k_{\max}$ is sufficiently large.

Returning to the task of convolution, we need to compute derivatives of

$$F_{\pm,\Delta,\ell}(a,b)=\left(\frac{(2-a)^2-b}{4}\right)^{\Delta_\phi}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(a,b)\pm\left(\frac{a^2-b}{4}\right)^{\Delta_\phi}g_{\Delta,\ell}^{\Delta_{12},\Delta_{34}}(2-a,b),\tag{2.21}$$

at $(a_*,b_*)=(1,0)$. We may immediately see that only one of the two terms in (2.21) needs to be differentiated. If the number of a derivatives is even (odd), the other term will contribute equally (oppositely) for $F_{+,\Delta,\ell}$ and oppositely (equally) for $F_{-,\Delta,\ell}$. We therefore reduce one vector of derivatives to another vector of derivatives having roughly half

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

the size. As in the unconvolved case, its components have the positive-times polynomial form. Knowing that Δ_ϕ will eventually be determined by the external dimensions $\Delta_i,\Delta_j,\Delta_k,\Delta_l$, we write

$$\frac{\partial^{m+n}}{\partial a^m \partial b^n} F^{ij;kl}_{\pm,\Delta,\ell}(a_*,b_*) = \chi_\ell(\Delta) P^{ij;kl;mn}_{\pm,\ell}(\Delta). \tag{2.22}$$

The linear combinations we need to take in order to compute these polynomials are known in closed form. For the following calculation, it is easiest to take all of the b derivatives first and then set $b=0$. This allows us to treat all terms as being linear in a

$$\begin{aligned}
&\frac{\partial^{m+n}}{\partial a^m \partial b^n}\left(\frac{(2-a)^2-b}{4}\right)^{\Delta_\phi} g_{\Delta,\ell} \\
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{n}{j}\frac{\partial^{i+j}}{\partial a^i \partial b^j}\left(\frac{(2-a)^2-b}{4}\right)^{\Delta_\phi}\frac{\partial^{m+n-i-j} g_{\Delta,\ell}}{\partial a^{m-i}\partial b^{n-j}} \\
&\to \sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{n}{j}\left(\frac{1}{4}\right)^j (-\Delta_\phi)_j, \qquad (2.23a)
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial^{i}}{\partial a^i}\left(1-\frac{a}{2}\right)^{2\Delta_\phi-2j}\frac{\partial^{m+n-i-j} g_{\Delta,\ell}}{\partial a^{m-i}\partial b^{n-j}} \\
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{n}{j}\left(\frac{1}{4}\right)^j\left(\frac{1}{2}\right)^i (-\Delta_\phi)_j (2j-2\Delta_\phi)_i, \qquad (2.23b)
\end{aligned}$$

$$\begin{aligned}
&\left(1-\frac{a}{2}\right)^{2\Delta_\phi-2j-i}\frac{\partial^{m+n-i-j} g_{\Delta,\ell}}{\partial a^{m-i}\partial b^{n-j}} \\
&\to \sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{n}{j}\left(\frac{1}{4}\right)^{\Delta_\phi} (-\Delta_\phi)_j (2j-2\Delta_\phi)_i \frac{\partial^{m+n-i-j} g_{\Delta,\ell}}{\partial a^{m-i}\partial b^{n-j}}. \qquad (2.23c)
\end{aligned}$$

We may now summarize how the input parameters d, $k_{\max}$, $\ell_{\max}$, $m_{\max}$, $n_{\max}$, Δ_{12}, Δ_{34} are used to prepare a conformal bootstrap environment. `PyCFTBoot`,

1. Creates a vector $\mathbf{h}_{\infty,\ell}$ containing r derivatives of (2.4) up to third order.
2. Calculates $\mathbf{d}_{\ell,i}$ residues with $k_{\max}$ iterations that use the data in (2.5) and Table 2.
3. Combines these into $\ell_{\max}$ meromorphic blocks $\mathbf{h}_{\Delta,\ell}$ through (2.11), optionally approximating small poles with (2.12).
4. Converts these into genuine conformal blocks $\mathbf{g}_{\Delta,\ell}$ with the matrix (2.13).
5. Applies the chain rule to get $\min(m_{\max}+2n_{\max},3)$ derivatives of all $g^{\Delta_{12},\Delta_{34}}_{\Delta,\ell}$ with respect to a.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

6. Uses (2.18), (2.19) to calculate whatever a, b derivatives are left and then uses (2.23) to take the convolution leaving Δ_ϕ as a free variable.

Almost all of the time is spent on the first four steps. The most interesting parameter here is $k_{\max}$ because these steps clearly have a running time which is sublinear in the number of spins. Table 3 times the calculation of a few increasingly accurate conformal block tables with the single spin $\ell=0$.

Table 3: Running time in seconds for the $d=3$ calculation of $g^{0,0}_{\Delta,0}(a,b)$ and its first three derivatives with respect to a. Three trials were done on one core of a 2.4GHz machine.

$k_{\max}$	Trial 1	Trial 2	Trial 3
10	1.106	1.119	1.111
15	3.146	3.151	3.127
20	6.859	6.856	6.873
25	13.731	13.844	13.682
30	23.062	23.058	23.131

2.3 Even dimensions

Unlike the exact expressions for conformal blocks [50, 51, 58] which are only known in even dimension, the scheme based on (2.10) that we have described so far works best when d is odd or fractional. This is because it assumes that all poles in Δ for a conformal block are simple. The breakdown of this assumption as ν becomes an integer can be seen as certain poles approach each other and certain residues diverge. In order to handle even dimensions, `PyCFTBoot` switches to a different method for calculating the initial conformal block derivatives. The idea is to change variables in (2.18) to get a differential equation in terms of r which may be solved by a power series. Since Δ is a solution of the indicial equation, the coefficients in

$$g^{\Delta_{12},\Delta_{34}}_{\Delta,\ell}(r,1)=r^\Delta\sum_{k=0}^{\infty}b_k r^k$$

may be determined recursively starting from $b_k=\delta_{k,0}$. Nothing stops us from applying this method in general dimension but we avoid doing so because it typically leads to polynomials of higher degree[¶].

The Frobenius method tells us to solve

$$k(\Delta+\ell+k-1)(\Delta-\ell+k-2\nu-1)(2\Delta+k-2\nu-2)b_k=\sum_{i=1}^{7}\gamma_{i,k}b_{k-i}. \tag{2.24}$$

[¶]It is nevertheless true that this method generates tables more quickly than (2.10) since it omits the $\eta\neq 1$ information. We thank Slava Rychkov for pointing this out.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

The radial version of the operator (2.18) needed for this step was given in [56]. The $\gamma_{i,k}$ were not so we now list them for completeness

$$\begin{aligned}
\gamma_{1,k} =& 2c_2(2\nu+1)(4S+1)-c_4+8P\nu(2\nu+1)\\
&-2(\Delta+k-1)[c_2(2\nu+1)+2P(6\nu-1)+8S(c_2+\nu-2\nu^2)]\\
&+2(\Delta+k-2)_2[c_2+\nu-2\nu^2+4P+12S(1-2\nu)]\\
&+2(\Delta+k-3)_3(2\nu-1+8S)-(\Delta+k-4)_4, \qquad (2.25a)\\
\gamma_{2,k} =& 3c_4-8P[2S(1-6\nu)+6\nu^2-5\nu]+2c_2[4S(4S+2\nu+1)+2\nu-3]\\
&-2(\Delta+k-2)[2P(16S-10\nu+4)+8S(c_2+\nu-2\nu^2)+(1-2\nu)(c_2+2\nu-2+32S^2)]\\
&-2(\Delta+k-3)_2[3c_2+7\nu+2\nu^2-10+4P+4S(10S+6\nu-3)]\\
&+2(\Delta+k-4)_3(7-2\nu+8S)+3(\Delta+k-5)_4, \qquad (2.25b)\\
\gamma_{3,k} =& 3c_4+2c_2(16S^2+2\nu-3)+16PS(8S+2\nu+5)\\
&-2(\Delta+k-3)[(1-2\nu)(c_2+2\nu-2+32S^2-4P)-8S(8S^2+4P+4\nu^2+2c_2-5)]\\
&-2(\Delta+k-4)_2[8P+40S^2+48S+3c_2+2\nu^2+7\nu-10]\\
&+2(\Delta+k-5)_3(7-2\nu-16S)+3(\Delta+k-6)_4, \qquad (2.25c)\\
\gamma_{4,k} =& 2c_2(3-2\nu-16S^2)-3c_4-16PS(8S+2\nu+5)\\
&-2(\Delta+k-4)\left[12+4\nu-8\nu^2-c_2(2\nu+5)\right.\\
&\left.-8S(8S^2+8S\nu+6S+4\nu^2+2c_2-5)+4P(2\nu-5-8S)\right]\\
&-2(\Delta+k-5)_2[3c_2+2\nu^2+7\nu-10+8P+S(40S-18\nu+21)]\\
&-2(\Delta+k-6)_3(16S+2\nu+11)-3(\Delta+k-7)_4, \qquad (2.25d)\\
\gamma_{5,k} =& 8P(2S(1-6\nu)+6\nu^2-5\nu)-3c_4-2c_2[4S(4S+2\nu+1)+2\nu-3]\\
&-2(\Delta+k-5)\left[12+4\nu-8\nu^2-c_2(2\nu+5-8S)\right.\\
&\left.+2P(3-10\nu+16S)-8S(2\nu^2+5\nu+3+8S\nu+6S)\right]\\
&-2(\Delta+k-6)_2[22+5\nu-2\nu^2-3c_2-4P-4S(10S+6\nu+9)]\\
&+2(\Delta+k-7)_3(8S-2\nu-11)-3(\Delta+k-8)_4, \qquad (2.25e)\\
\gamma_{6,k} =& c_4-2c_2(2\nu+1)(4S+1)+8P\nu(2\nu+1)\\
&-2(\Delta+k-6)[(2\nu+3)(c_2-2\nu-2)+6P(2\nu+1)+8S(c_2-2\nu^2-5\nu-3)]\\
&-2(\Delta+k-7)_2[c_2+4P-(2\nu+3)(\nu+4+12S)]\\
&+2(\Delta+k-8)_3(2\nu+5+8S)+(\Delta+k-9)_4, \qquad (2.25f)\\
\gamma_{7,k} =& (k+2\nu-5)(2\Delta+k-7)(\Delta+k-\ell-6)(\Delta+k+\ell+2\nu-6). \qquad (2.25g)
\end{aligned}$$

In order to continue the practice of storing numerators and denominators separately, we use (2.24) to calculate b_k coefficients multiplied by polynomials of degree $3k$. After this, the second last coefficient must be multiplied by three factors that are only in the last one. The third last coefficient must be multiplied by six factors that are only in the last two,

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

etc.. The case $P=S=0$ is especially simple because we only have to use $\gamma_{2,k}$, $\gamma_{4,k}$ and $\gamma_{6,k}$ when our conformal block is an expansion in r^2. In the rest of this section, we take a closer look at how meromorphic conformal blocks remain finite as d approaches an even integer. This is independent of the rest of the paper since there is no code in `PyCFTBoot` that makes use of it[‖].

From (2.5), we see that only $c_2^{\Delta_{12},\Delta_{34}}(\ell,k)$ can ever be infinite. This reflects the fact that a pair of coincident poles in Table 2 must always involve series 2. Problematic terms where equal poles are subtracted may cancel in one of two ways:

1. A term like this that multiplies an expression with Δ may combine with an infinite residue that multiplies a similar expression with Δ. Consider

$$\frac{1}{\{\nu\}}\frac{1}{\Delta-\Delta_1}-\frac{1}{\Delta-\Delta_2}\frac{1}{\Delta_2+n-\Delta_3},$$

   where we have split $\nu=\lfloor\nu\rfloor+\{\nu\}$ into its integer and fractional part. If $\Delta_2=\Delta_1-\{\nu\}$ and $\Delta_3=\Delta_2+n-\{\nu\}$, we may rewrite this as

$$\frac{1}{\{\nu\}}\left(\frac{1}{\Delta-\Delta_1}-\frac{1}{\Delta-\Delta_1+\{\nu\}}\right)=\frac{1}{(\Delta-\Delta_1)(\Delta-\Delta_1+\{\nu\})},$$

   which has a finite limit.

2. The residue being divided by a difference of equal poles might be proportional to $\{\nu\}$ itself.

To see the first type of cancellation, we may set Δ_{12}, Δ_{34}, ℓ and $\lfloor\nu\rfloor$ to zero. In this case $h_{\infty,\ell}(r,1)=\frac{1}{1-r^2}$. Going up to r^4,

$$\begin{aligned}h_{\Delta,0}&=\frac{1}{1-r^2}+\frac{c_1(0,2)r^2}{\Delta+1}h_{1,2}+\frac{r^4}{1-r^2}\left[\frac{c_1(0,4)}{\Delta+3}+\frac{c_2(0,2)}{\Delta+1-\nu}\right]\\&=\frac{1}{1-r^2}+\frac{r^2}{1-r^2}\frac{c_1(0,2)}{\Delta+1}+\frac{r^4}{1-r^2}\frac{c_1(0,2)}{\Delta+1}\left(\frac{c_1(2,2)}{4}-\frac{c_3(2,2)}{2\nu}\right)\\&\quad+\frac{r^4}{1-r^2}\left[\frac{c_1(0,4)}{\Delta+3}+\frac{c_2(0,2)}{\Delta+1-\nu}\right].\end{aligned}\tag{2.26}$$

We may now focus on what is proportional to r^4. Terms in square brackets come from the first level of the recurrence relation while terms in round brackets come from the second. Taking one of each, we may form the combination

$$\frac{c_2(0,2)}{\Delta+1-\nu}-\frac{c_3(2,2)}{2\nu}\frac{c_1(0,2)}{\Delta+1}=\frac{1}{4\nu}\left(\frac{1}{\Delta+1-\nu}-\frac{1}{\Delta+1}\right)=\frac{1}{4(\Delta+1)(\Delta+1-\nu)}.\tag{2.27}$$

[‖] An especially slow piece of code for this used to be present. Interested readers may find it in the program's `git` history.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

If all divergences were to cancel in this way, it would make sense to ignore all elements of (2.5) that are 0 or ∞ and infer their effects later on. For instance, when two poles meant to be subtracted in the denominator coincide, this can be taken as a signal to instead square the pole difference that exists one level up. Unfortunately, because the second type of cancellation is common as well, a robust implementation would have to keep all residues and temporarily equip them with a free symbol for $\{\nu\}$. A 4D recursion to order r^6 shows the other phenomenon

$$h_{\Delta,0}=h_{\infty,0}+\frac{r^2c_1(0,2)}{\Delta+1}h_{1,2}+\frac{r^2c_2(0,1)}{\Delta-1}h_{3,0}+\frac{r^4c_1(0,4)}{\Delta+3}h_{1,4}+\frac{r^4c_2(0,2)}{\Delta}h_{4,0}$$
$$+\frac{r^6c_1(0,6)}{\Delta+5}h_{\infty,6}+\frac{r^6c_2(0,3)}{\Delta+1}h_{\infty,0}. \tag{2.28}$$

The term in red is infinite and needs to be cancelled by something. This tells us to look at the blue term because it also includes a $\frac{1}{\Delta+1}$. Expanding this meromorphic block and not setting its dimension to 1 yet,

$$h_{\Delta,2}=h_{\infty,2}+\frac{r^2c_1(2,2)}{\Delta+3}h_{-1,4}+\frac{r^2c_2(2,1)}{\Delta-1}h_{3,2}+\frac{r^2c_3(2,2)}{\Delta-3}h_{5,0}$$
$$+\frac{r^4c_1(2,4)}{\Delta+5}h_{-1,6}+\frac{r^4c_2(2,2)}{\Delta}h_{4,2}. \tag{2.29}$$

The term in magenta cannot be ignored. Even though $c_2(2,1)$ vanishes with $\{\nu\}$, so does $\Delta-1$ once we substitute the dimension. Using the fact that this is finite to plug (2.29) into itself one more time, we see that the $\Delta-3$ term provides the next divergence. This is what gives the blue term a divergence two levels up allowing it to cancel the red one. Since double poles appear at all levels of the recursion, algebraic simplifications need to be performed repeatedly, slowing down the calculation. Moreover, they only work correctly if all terms are placed over a common denominator–not just the ones with the free variable Δ. This causes exponentially large numerators and denominators to accumulate during the calculation of $\mathbf{d}_{\ell,i}$ even when the fractions themselves are small. Neglecting the error introduced by this would require many more digits than those kept by [11, 23] and other high precision studies.

3 Overall structure

We now describe the three objects in a typical `PyCFTBoot` session that involve tables of polynomials in Δ. These include an object for the semidefinite program itself which is most directly relevant for the user. The steps described so far ending with convolution amount to two lines of code with mostly self-explanatory arguments

```
table1 = ConformalBlockTable(dim, k_max, l_max, m_max, n_max,
          delta_12, delta_34, odd_spins = True)
table2 = ConvolvedBlockTable(table1, symmetric = True)
```

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

Table 4: These global variables of `PyCFTBoot` are symbols in the sense that they are treated as variables for the computer algebra.

Symbol	Description
`delta`	The scaling dimension variable on which all polynomials depend.
`delta_ext`	A placeholder for Δ_ϕ in `ConvolvedBlockTable`.
`ell`	A variable for the few situations that need an unspecified spin.

A slower version of `ConformalBlockTable` which should almost never be needed is `ConformalBlockTableSeed` (or `ConformalBlockTableSeed2` in even dimension). Although it is meant to be used internally to prepare a (2.18) recursion by calculating the first three derivatives, it can be used to calculate more derivatives explicitly as well. Global variables affecting these two lines of code are `prec` and `cutoff`. The default value of `prec` (the binary precision) is 660, consistent with the 200 decimal digits of `SDPB`'s example code. The θ variable in (2.12) is `cutoff` which must be set manually if the user wants it to differ from 0. Another variable which should be set manually is `sdpb_path` unless the default value of `/usr/bin/sdpb` is correct. In addition to these global variables, there are global symbols defined in Table 4.

Two optional parameters have been set to `True`. For `odd_spins`, this indicates that odd spins from 0 to ℓ_{max} should not be skipped. For `symmetric`, it indicates that $F_{+,\Delta,\ell}$ is being calculated rather than the default $F_{-,\Delta,\ell}$. There are two other optional parameters that could have been passed above. For `ConformalBlockTable`, the `name` parameter tells it to ignore all other arguments, avoid doing any calculation and instead prepare a conformal block table by reading a file. These files are generated by calling `table1.dump("filename")`. For `ConvolvedBlockTable`, the `content` parameter tells the class to produce a linear combination of convolved conformal blocks with prescribed coefficients if the operators are part of a larger (e.g., superconformal) multiplet. The elements of this list need further explanation. If one term in the linear combination is a regular convolved block, a subsequent term is specified by three things: an expression for the coefficient, a number indicating how different its Δ is and an integer indicating how different its ℓ is. An artificial example is a multiplet which has conformal blocks (and hence convolved conformal blocks) arranged as follows:

$$\mathcal{G}_{\Delta,\ell} = \frac{1}{\Delta+\ell} g_{\Delta,\ell} + \Delta g_{\Delta-1,\ell+1}, \tag{3.1a}$$

$$\mathcal{F}_{\pm,\Delta,\ell} = \frac{1}{\Delta+\ell} F_{\pm,\Delta,\ell} + \Delta F_{\pm,\Delta-1,\ell+1}. \tag{3.1b}$$

In this case, one needs to multiply everything by $\Delta+\ell$ to avoid breakage due to non-polynomial terms. Afterwards, this two element linear combination where each term has three pieces of data, is passed as a pair of triples. One simply gives

```
content = [[1, 0, 0], [delta * (delta + ell), -1, 1]]
```

to `ConvolvedBlockTable`.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

3.1 Working with SDPs

The final class to discuss, the `SDP`, specifies the arrangement of convolved conformal blocks that needs to vanish for crossing symmetry to hold. The fundamental objects for these sum rules are

$$F^{ij;kl}_{\pm,\Delta,\ell}(u,v)=v^{\frac{\Delta_j+\Delta_k}{2}}g^{\Delta_{ij},\Delta_{kl}}_{\Delta,\ell}(u,v)\pm u^{\frac{\Delta_j+\Delta_k}{2}}g^{\Delta_{ij},\Delta_{kl}}_{\Delta,\ell}(v,u). \tag{3.2}$$

The Δ_ϕ variable in (2.21) may be replaced with all possible values of $\frac{\Delta_j+\Delta_k}{2}$ that can be made from the correlator system under consideration. Linear combinations of the (3.2) blocks need to give zero in all crossing equations. The weights for these are built out of OPE coefficients which are real by unitarity. When $i=j=k=l$, we simply have squares of OPE coefficients which are positive. In this case, the equation $\sum_{\mathcal{O}}\lambda^2_{\mathcal{O}}F_{-,\Delta,\ell}(u,v)=0$ rules out a CFT whenever some functional Λ is positive on all $F_{-,\Delta,\ell}$. In more complicated cases, we do not necessarily have positive coefficients. One example [24] is the crossing equation with no global symmetry:

$$\sum_{\mathcal{O}}\left[\lambda_{ij\mathcal{O}}\lambda_{kl\mathcal{O}}F^{ij;kl}_{\mp,\Delta,\ell}(u,v)\pm\lambda_{kj\mathcal{O}}\lambda_{il\mathcal{O}}F^{kj;il}_{\mp,\Delta,\ell}(u,v)\right]=0. \tag{3.3}$$

Here, it is not useful to find a Λ sending all $F^{ij;kl}_{\pm,\Delta,\ell}$ to a positive number. What we must do is find a Λ that sends particular groupings of them to a positive definite matrix. This is an example of a polynomial matrix problem (a special case of semidefinite program)

$$\begin{aligned}&\text{maximize} && \Lambda\cdot o\\ &\text{such that} && \Lambda\cdot P_{\ell,R}(x)\succeq 0 \text{ for all } x\geq 0,\ell,R,\\ &&& \Lambda\cdot n=1,\end{aligned} \tag{3.4}$$

which is what `SDPB` solves. The objective o, the normalization n and the exact relation between x and Δ are not needed to initialize an `SDP` class. However, the representations R and the groupings of blocks mentioned above need to be passed in a parameter. Let us call this parameter `info` and imagine that our correlator system has two operators σ and ϵ with $(\Delta_\sigma,\Delta_\epsilon)=(0.7,1.5)$. If `table3` is another `ConvolvedBlockTable` instance like `table2` above, we may call

```
sdp = SDP([0.7, 1.5], [table2, table3], vector_types = info)
```

to get a new `SDP`. The tables and dimensions above may be specified in an arbitrary order but indices describing their positions in the list are obtained from `info`. The `vector_types` argument is required unless both of the first two arguments are single elements. Suppose that the sum in (3.3) runs over one representation and all spins. Even and odd spins are considered separately so from the point of view of `PyCFTBoot`, this leads to two representations A and B which we label with `0` and `1` respectively

```
info = [[info1, 2, 0], [info2, 3, 1]].
```

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

The 2 and 3 have been chosen because any even integer denotes even spin and any odd integer denotes odd spin. Note that `info = [[info2, 3, 1], [info1, 2, 0]]` would be wrong because the first representation must be the one containing the identity operator. Now suppose that there are three crossing equations with 2×2 matrices in the A parts and 1×1 matrices in the B parts. They might look something like

$$\sum_{\mathcal{O}\in A}(\lambda_{\sigma\sigma\mathcal{O}}\lambda_{\epsilon\epsilon\mathcal{O}})\begin{bmatrix}\begin{pmatrix}0 & \frac{1}{2}F^{\sigma\sigma;\sigma\sigma}_{-,\Delta,\ell}\\ \frac{1}{2}F^{\sigma\sigma;\sigma\sigma}_{-,\Delta,\ell} & 0\end{pmatrix}\\ \begin{pmatrix}F^{\sigma\sigma;\sigma\sigma}_{-,\Delta,\ell} & 0\\ 0 & F^{\sigma\sigma;\sigma\sigma}_{-,\Delta,\ell}\end{pmatrix}\\ \begin{pmatrix}0 & \frac{1}{2}F^{\epsilon\epsilon;\epsilon\epsilon}_{-,\Delta,\ell}\\ \frac{1}{2}F^{\epsilon\epsilon;\epsilon\epsilon}_{-,\Delta,\ell} & 0\end{pmatrix}\end{bmatrix}\begin{pmatrix}\lambda_{\sigma\sigma\mathcal{O}}\\ \lambda_{\epsilon\epsilon\mathcal{O}}\end{pmatrix}$$
$$+\sum_{\mathcal{O}\in B}\lambda^2_{\sigma\epsilon\mathcal{O}}\begin{bmatrix}0\\ F^{\sigma\epsilon;\sigma\epsilon}_{+,\Delta,\ell}\\ \frac{3}{2}F^{\sigma\epsilon;\sigma\epsilon}_{+,\Delta,\ell}\end{bmatrix}=0. \tag{3.5}$$

Triples of matrices are easy to specify with Python but each matrix element is encoded by four pieces of information: a real coefficient, an integer labelling the convolved conformal block and integers labelling the inner two (j,k in the (3.2) notation) dimensions. If our `SDP` is applicable to this system, its `[table2, table3]` list contains one symmetric convolved block with $\Delta_{\sigma\epsilon}$ differences and one antisymmetric convolved block with 0 differences. If they appear in this order, the innermost lists of `info1` have a `1` in the second position while those of `info2` have a `0`. Indeed, one may check that

```
info1 =[[[[0.0, 1, 0, 0], [0.5, 1, 0, 0]], [[0.5, 1, 0, 0], [0.0, 1, 0, 0]]],
        [[[1.0, 1, 0, 0], [0.0, 1, 0, 0]], [[0.0, 1, 0, 0], [1.0, 1, 0, 0]]],
        [[[0.0, 1, 0, 0], [0.5, 1, 1, 1]], [[0.5, 1, 1, 1], [0.0, 1, 0, 0]]]],
info2 =[[0.0, 0, 0, 0], [1.0, 0, 0, 1], [1.5, 0, 0, 1]],
```

fully describes this artificial example.

Before we describe the various ways in which `SDPB` can be called to do the heavy lifting, it is useful to explore the structure of the allocated `SDP`. Most of the memory is occupied by `table`, a three-dimensional list storing the polynomials in Δ. The first index runs over operators from the (3.3) sum rule meaning spins and representations. The second and third indices label elements of the matrices that must become positive definite under Λ. These come in the order given by `vector_types`. In (3.5), consider two indices `a` and `b` on either side of the "middle" `len(sdp.table) / 2` element. Since `a` corresponds to an A operator, it is perfectly valid for the user to type `sdp.table[a][0][1]` or `sdp.table[a][1][0]`. With B operators however, only `sdp.table[b][0][0]` is a valid query. Elements thus returned correspond to infinite-dimensional functions, but we have

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

already gone to great lengths to approximate each of these with a finite-dimensional vector of derivatives evaluated at the crossing symmetric point. The object storing this type of truncation is called a `PolynomialVector`. It has three attributes of which `vector` is the most important. This is what stores the actual polynomials in Δ. They are essentially the convolved conformal block polynomials from (2.22) except they have been multiplied by the appropriate coefficients in `vector_types`. The length of something like `sdp.table[b][0][0].vector` depends on $m_{\max}$ and $n_{\max}$ since each element is a derivative. However, derivatives are often repeated when the sum rule is vectorial. Looking at (3.5), instead of simply seeing infinite-dimensional functions, each term is a *triple* of infinite-dimensional functions. `PyCFTBoot` concatenates the derivatives used to approximate each one. This makes it difficult to remember what each polynomial represents. For instance, if we were naive enough to include no b derivatives, `sdp.table[b][0][0].vector` would be

$$\left[0,0,0,P_{+,b}^{\sigma\epsilon;\sigma\epsilon;00}(\Delta),P_{+,b}^{\sigma\epsilon;\sigma\epsilon;20}(\Delta),P_{+,b}^{\sigma\epsilon;\sigma\epsilon;40}(\Delta),\frac{3}{2}P_{+,b}^{\sigma\epsilon;\sigma\epsilon;00}(\Delta),\frac{3}{2}P_{+,b}^{\sigma\epsilon;\sigma\epsilon;20}(\Delta),\frac{3}{2}P_{+,b}^{\sigma\epsilon;\sigma\epsilon;40}(\Delta)\right]^{\mathrm{T}}. \tag{3.6}$$

The `SDP` type includes two lists that remind us of where different derivatives are positioned. To see how many a and b derivatives are encoded by a given element, one only needs to check the corresponding elements of `sdp.m_order` and `sdp.n_order` respectively. The two other attributes of a `PolynomialVector`–`poles` and `label`–are also Python lists. The elements of `poles` are the poles from Table 2 that must be used to reconstruct the positive prefactor of (2.22). The `label` is a two element list with a spin first and a representation label second[**]. One more interesting attribute is `sdp.unit`, the contribution of the identity. This is the one operator that is guaranteed to appear in every crossing equation. To calculate this, `SDP` substitutes $\Delta=\ell=0$ into the `table` elements that have R as the singlet representation. It also multiplies by the proper OPE coefficients. These are known because the canonically normalized

$$\langle\phi_i(x_1)\phi_j(x_2)\rangle=\frac{\delta_{ij}}{|x_{12}|^{\Delta_i+\Delta_j}}, \tag{3.7a}$$

$$\langle\phi_i(x_1)\phi_j(x_2)\phi_k(x_3)\rangle=\frac{\lambda_{ijk}}{|x_{12}|^{\Delta_i+\Delta_j-\Delta_k}|x_{23}|^{\Delta_j+\Delta_k-\Delta_i}|x_{13}|^{\Delta_k+\Delta_i-\Delta_j}}, \tag{3.7b}$$

are only consistent with each other if all $\lambda_{ijI}=1$.

When numerically excluding CFTs, the most obvious physical inputs are the allowed ranges for the scaling dimensions in a trial spectrum. When $\Delta\in[\Delta_{\min},\infty)$, all polynomials should have Δ replaced by $\Delta_{\min}+x$ so that x satisfies the positivity requirement in (3.4).

[**]The `table` attributes of `ConformalBlockTable` and `ConvolvedBlockTable` have very similar layouts. Because no vectors of matrices are present at this stage, no derivatives are repeated in the `PolynomialVectors` and only one index is needed to iterate over them. Inspecting their `label` attributes, we see that the second element is always `0`. This is because no other labels have been given in `vector_types` yet.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

In a CFT that is unitary but otherwise unconstrained, Δ_{min} is equal to the unitarity bound,

$$\Delta_{\text{unitary}} = \begin{cases} \frac{d-2}{2}, & \ell = 0, \\ d+\ell-2, & \ell > 0. \end{cases} \tag{3.8}$$

Although bounds for the `SDP` class are always initialized to (3.8), they may be changed with a call to `sdp.set_bound([l, r], delta_min)`. Here `l` is a spin and `r` is a representation label. These bounds continue to be enforced until they are undone manually. Resetting a given (ℓ,R) to the unitarity bound is done with `sdp.set_bound([l, r])`. Omitting both arguments causes `PyCFTBoot` to reset the bounds of all operators. In the exact same manner, individual points may be added with `sdp.add_point([l, r], delta_value)`. These are explicitly allowed dimensions at which `PolynomialVectors` should be evaluated. Calling `sdp.add_point([0, 0], 1.0)` prepares us for bootstrapping a theory with spin-0 singlets of dimension 1, even after something like `sdp.set_bound([0, 0], 1.2)` has been called. Again, removing points for a given operator type or all operator types may be accomplished by omitting arguments. The last persistently stored property of an `SDP` is the list of options passed to `SDPB`. The options that `PyCFTBoot` correctly chooses without user interaction are `--precision` and all options not passed as key-value pairs. For everything else, a helper function is provided. As an example, one may leave some processor resources unused by passing the key-value pair `--maxThreads=2`. `PyCFTBoot` can be told to use this with the method `sdp.set_option("maxThreads", 2)`. The line undoing this is `sdp.set_option("maxThreads")` and the line undoing everything is `sdp.set_option()`.

3.2 Writing XML files

`SDPB` learns everything that it needs to know about an optimization from an XML file [23]. Knowing that the points and bounds determine x in (3.4) while `sdp.table` determines the polynomials, only the objective o and the normalization n are needed to write the XML. To rule out CFTs with a certain gap, one chooses an objective vector of zero and a normalization of `sdp.unit`. Another common task is maximizing a squared OPE coefficient. For this, the objective vector must be `sdp.unit` with the `PolynomialVector` for the (Δ,ℓ,R) being maximized as the normalization. These are specified using `sdp.write_xml(obj, norm, "name")` but it is often not necessary to call this function directly. A more convenient function is an implementation of the bisection described in [4], which works for identical scalars. To bisect over gaps in an (ℓ,R) operator, one should call `sdp.bisect(lower, upper, tol, [l, r])`. Since this method finds upper bounds, `upper` should be a gap where a Λ solving (3.4) exists and `lower` should be a gap where such a Λ does not. The boundary between allowed and disallowed regions is returned with a tolerance of `tol`. Since points on either side of the boundary take different amounts of time to test, it is convenient to use a biased binary search. Once

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

a long test taking time L and a short test taking time S have been performed, we may approximate the optimal bias β by minimizing the time:

$$T=\#(\beta)[\beta L+(1-\beta)S]\propto\frac{\beta L+(1-\beta)S}{\log\left[\beta^{\beta}(1-\beta)^{(1-\beta)}\right]}. \tag{3.9}$$

We have used the fact that β is also the probability that a given test will reduce our uncertainty by a factor of β. Strictly speaking, when `SDPB` finishes finding a functional, the problem is called primal-dual optimal (another word for "primal and dual feasible"). The bisection in `PyCFTBoot` does not wait for this to happen. Rather, it takes advantage of a very safe assumption: when dual feasibility (time L) is achieved before primal feasibility (time S) during `SDPB`'s iterations, it is only a matter of time before primal feasibility is achieved as well[††]. If the full solution functional is desired after a bisection, it may be found with `sdp.solution_functional(gap, [l, r])`. For an excluded Δ, the returned functional must turn (ℓ,R)'s matrix of `PolynomialVectors` into something positive definite. It is therefore useful to tune Δ until the determinant of this matrix is exactly zero. These zeros, returned by `sdp.extremal_dimensions(functional, [l, r])`, are exactly the scaling dimensions in the spectrum of a CFT that lives on the boundary [57]. Rather than obeying $\Lambda\cdot n=1$, the functionals used by these methods are normalized to have a leading component of 1. This reflects the alternate definition of a semidefinite program used by `SDPB`. Instead of (3.4), the program solves

$$\begin{aligned}&\text{maximize} && \Lambda\cdot\tilde{o}\\ &\text{such that} && \Lambda\cdot\tilde{P}_{\ell,R}(x)\succeq 0 \text{ for all } x\geq 0,\ell,R,\\ &&& \Lambda_0=1,\end{aligned} \tag{3.10}$$

which is trivially equivalent [23]. One simply substitutes

$$\Lambda_0=\frac{1}{n_0}\left(1-\sum_{i=1}^{N}\Lambda_i n_i\right)$$

into (3.4). When we once again collect all terms proportional to Λ_i, we find that the i^{th} component of the reshuffled $\tilde{P}_{\ell,R}$ involves components i and 0 of $P_{\ell,R}$. Finally, we should describe `PyCFTBoot`'s built-in method for bounding OPE coefficients. The logic for this is easiest to write in the single correlator case: $\sum_{\mathcal{O}\neq I}\lambda^2_{\mathcal{O}}F_{-,\mathcal{O}}=-F_{-,I}$. Normalizing Λ on the convolved conformal block of some particular $\mathcal{O}'$ [6],

$$\lambda^2_{\mathcal{O}'}=\Lambda(F_{-,I})-\sum_{\mathcal{O}\neq I,\mathcal{O}'}\lambda^2_{\mathcal{O}}\Lambda(F_{-,\mathcal{O}})\leq\Lambda(F_{-,I}). \tag{3.11}$$

[††]One must be careful when assuming the converse: that only unsolvable problems will achieve primal feasibility first. Sometimes when testing a point far from the boundary, a primal-dual optimal solution will be approached with the opposite order.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

The last step of neglecting the strictly positive terms is still valid in the multi-correlator case. However, if multiple OPE coefficients in the crossing equations involve the operator $\mathcal{O}'$, the left hand side of (3.11) will involve the quadratic form $\boldsymbol{\lambda}_{\mathcal{O}'}^{\mathrm{T}}\Lambda(M_{\mathcal{O}'})\boldsymbol{\lambda}_{\mathcal{O}'}$. In this case, only the length of the OPE coefficient vector will be bounded. Since Λ turns the vector of matrices $M_{\mathcal{O}'}$ into a single positive definite matrix, we simply use the fact that the inequality is preserved if we replace this matrix with the identity multiplied by its smallest eigenvalue. This technique of using $\Lambda(F_{-,I})$ to bound a squared OPE coefficient is implemented by `sdp.opemax(delta_value, [l, r])`. Here, (Δ,ℓ,R) are the quantum numbers of $\mathcal{O}'$.

Users may notice a time delay when allocating an `SDP` class. This is used for a calculation involving the positive $\chi_\ell(\Delta)$ prefactors in (2.22) which have been largely ignored up to this point. Even though the semidefinite program itself is not affected, incorporating these into the XML file can significantly improve the performance and numerical stability of `SDPB` [23]. The non-trivial step is the calculation of a bilinear basis–a set of polynomials that are orthogonal with respect to the $\chi_\ell(\Delta_{\min}+x)$ measure on $(0,\infty)$. Since these functions only change when the bounds change, the bases do not need to be recalculated every time an XML file is written. After `PyCFTBoot` allocates an `SDP` and calculates all bases, a particular ℓ's basis is only recalculated when `sdp.set_bound([l, r])` is called. This saves time during a bisection because the many XML files generated only have different bounds for a single operator type. Multi-correlator bootstraps, which cannot use bisection, typically have all of their XML files generated by different instances of `SDP`. To avoid a performance hit in this case, `PyCFTBoot` provides an optional argument to `SDP` called `prototype`. This allows an existing `SDP` to have its bilinear basis recycled in the allocation of a new one. For completeness, we now review the most direct method for finding $m+1$ polynomials orthogonal under

$$\chi_\ell(x)=\frac{r_*^{x+\Delta_{\min}(\ell)}}{\prod_i(x+\Delta_{\min}(\ell)-\Delta_i(\ell))}. \tag{3.12}$$

We may clearly multiply by $r_*^{\Delta_{\min}}$ and shift the poles so we will omit $\Delta_{\min}$ in what follows. What we are trying to find is the matrix

$$L=\begin{bmatrix} q_{00} & 0 & 0 & \cdots & 0 \\ q_{10} & q_{11} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{m0} & q_{m1} & q_{m2} & \cdots & q_{mm} \end{bmatrix}, \tag{3.13}$$

where the i^{th} polynomial is $q_i(x)=q_{i0}+q_{i1}x+\cdots+q_{ii}x^i$. The statement of orthonormality is

$$\int_0^\infty L\begin{bmatrix} 1 \\ \vdots \\ x^m \end{bmatrix}[1\cdots x^m]L^{\mathrm{T}}\chi(x)\mathrm{d}x=I. \tag{3.14}$$

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

If we let $M_{ij}=\langle x^i,x^j\rangle$ elements come from the positive definite matrix of inner products, (3.14) says that $LML^{\mathrm{T}}=I$. Since $M=L^{-1}L^{-1\mathrm{T}}$, L is the inverse of the lower triangular matrix in the Cholesky decomposition of M [59]. M has anti-diagonal bands because M_{ij} is fully determined by $i+j$. Therefore, our problem reduces to the evaluation of $2m+1$

$$\begin{aligned}\int_0^\infty \frac{x^n r_*^x}{\prod_i(x-\Delta_i)}\mathrm{d}x &= \sum_i \frac{1}{\prod_{j\neq i}(\Delta_j-\Delta_i)}\int_0^\infty \frac{x^n r_*^x}{x-\Delta_i}\mathrm{d}x \\ &= \sum_i \frac{1}{(-\log r_*)^n\prod_{j\neq i}(\Delta_j-\Delta_i)}\int_0^\infty \frac{y^n e^{-y}}{y+\Delta_i\log r_*}\mathrm{d}y \end{aligned} \tag{3.15}$$

integrals. Above, we have used a partial fraction decomposition valid for simple poles and made the substitution $y=-x\log r_*$. Allowing for incomplete gamma functions, the integral in (3.15) may be evaluated explicitly. This is because the integral representation

$$\Gamma(-n,z)=\frac{z^{-n}e^{-z}}{\Gamma(1+n)}\int_0^\infty \frac{y^n e^{-y}}{y+z}\mathrm{d}y \tag{3.16}$$

follows from the relation between the upper incomplete gamma function and the ${}_1F_1$ hypergeometric function. A related expression which may arise in even d is

$$\int_0^\infty \frac{y^n e^{-y}}{(y+z)^2}\mathrm{d}y = nz^{n-1}e^z\Gamma(n)\left[\Gamma(1-n,z)-z\Gamma(-n,z)\right]. \tag{3.17}$$

This follows from partial integration. By handling the slightly more complicated partial fraction decomposition, which `PyCFTBoot` and the `SDPB` example code can easily do, this allows us to still find M when χ_ℓ has double poles. We should note that the Cholesky decomposition of M can only be found when all of its eigenvalues are significantly greater than 0. Even if the desired polynomial precision is fairly low, this requirement forces us to keep many digits before the polynomials can be found at all. It would be interesting to see if indirect polynomial algorithms [60] can make this step less demanding.

4 Some examples

This section contains longer code snippets to show how the main tasks in the conformal bootstrap can be accomplished. It can also be used for reference since examples are often preferable to more verbose documentation. `PyCFTBoot` is four files at the time of writing but importing `bootstrap.py` will automatically import the others. They should be placed in the working directory or one of the system directories searched by Python.

4.1 Identical scalars

We begin with the simplest bound: the dimension of ϕ^2 in terms of the dimension of ϕ. There must be a $\mathbb{Z}_2$ symmetry for these to be different. To set this up in `PyCFTBoot`, we

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

run:

```
>>> from bootstrap import *
>>> table1 = ConformalBlockTable(3, 15, 15, 1, 3)
>>> table2 = ConvolvedBlockTable(table1)
```

Recall that this sets up a $d=3$ table with 15 poles, 15 spins and a 1×3 triangle of derivatives. If we are interested in the Δ_{ϕ^2} close to where the Ising model is known to live, we should run:

```
>>> sdp = SDP(0.52, table2)
>>> result = sdp.bisect(0.7, 1.7, 0.01, 0)
>>> result
1.434375
```

Although we are limiting ourselves to an error of 1%, the error is more likely dominated by missing derivatives. The last argument of `0` (`[0, 0]` is only necessary when there is more global symmetry) states that we are bounding scalars in the $\phi\times\phi$ OPE rather than operators of higher spin.

4.2 A faster approach

This bound may be bisected more quickly if we use [10]'s method to only increase the polynomial degree for sufficiently large residues. Since the `cutoff` controlling this is a global variable, it can only be altered if we keep it in its own namespace

```
>>> import bootstrap
>>> bootstrap.cutoff = 1e-10
>>> table1 = bootstrap.ConformalBlockTable(3, 15, 15, 1, 3)
>>> table2 = bootstrap.ConvolvedBlockTable(table1)
>>> sdp = bootstrap.SDP(0.52, table2)
>>> result = sdp.bisect(0.7, 1.7, 0.01, 0)
>>> result
1.434375
```

The output from `SDPB` (supressed above) shows that the iterations take less time and that they are slightly fewer in number.

4.3 The next scalar

The results above state that consistent CFTs at $\Delta_\phi=0.52$ stop existing once the scalar part of the search space starts at 1.44. The search space is allowed to be both discrete and

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

continuous, so even if it starts at 1.434375, we may still require that gaps for all other operators are significantly greater. Beginning in the same way as before,

```
>>> sdp = SDP(0.52, table2)
>>> sdp.add_point(0, 1.434375)
>>> result = sdp.bisect(1.44, 8.0, 0.01, 0)
>>> result
4.8225
```

where we have taken the extra step of adding a point. This tells us that one possible CFT living near the edge of the Δ_{ϕ^2} bound has 4.8225 as the dimension of its next $\mathbb{Z}_2$-even scalar. However, spectra saturating these bounds are unique [57]. The extremal functional method exploits this fact to find several low-lying dimensions at once without having to repeat the above procedure. To use this in `PyCFTBoot`, we need to set Δ_{ϕ^2} to something slightly larger than 1.434375 where crossing symmetry holds. By our bisection, the closest value where an extremal functional exists is at most 0.01 more than this

```
>>> sdp = SDP(0.52, table2)
>>> func = sdp.solution_functional(1.434375 + 0.01, 0)
>>> spec = sdp.extremal_dimensions(func, 0)
>>> spec
[0.95491336461809961, 1.4442910577901338, 4.3268882750844018]
```

The `0` arguments above are again short for (ℓ, R) labels of `[0, 0]`. The three dimensions returned in the spectrum include an inadmissible value, the bound we imposed and the desired second scalar. This time, it appears much closer to the high precision estimate obtained in [11].

4.4 Imposing another gap

The previous example shows us how to fix ϕ^2 and then constrain higher scalars in $\phi \times \phi$. It is often just as useful to proceed in the other direction. For example, the allowed region in $(\Delta_\phi, \Delta_{\phi^2})$ space shrinks if we first make additional assumptions on the second $\mathbb{Z}_2$-even scalar. This is particularly natural in the Ising model, which constrains this operator to be irrelevant by definition. We need to increase the derivative order and kept pole order

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

to see a noticeable effect:

```
>>> import bootstrap
>>> bootstrap.cutoff = 1e-10
>>> sig = 0.52
>>> eps = 1.42
>>> table1 = bootstrap.ConformalBlockTable(3, 20, 15, 2, 4)
>>> table2 = bootstrap.ConvolvedBlockTable(table1)
```

It only makes sense to bisect when the boundary of the allowed region is the graph of some function. This is no longer the case when we demand that the dimensions of two internal operators are a certain distance apart.

```
>>> sdp = bootstrap.SDP(sig, table2)
>>> sdp.set_bound(0, 3.0)
>>> sdp.add_point(0, eps)
>>> result = sdp.iterate()
>>> result
True
```

Clearly a point close to the Ising model is still allowed. However, if we start with `(sig, eps) = (0.52, 1.2)` and keep the rest of the code the same, it can easily be seen that `sdp.iterate()` returns `False` and begins to reveal a non-trivial shape. A repeated scan over many different values is what produces the plot in [9] with a sharp corner.

4.5 OPE maximization

Non-trivial features also appear in OPE coefficient bounds. As explained in (3.11), `PyCFTBoot` has a method for dealing with this part of the CFT data as well. Using the same setup as before,

```
>>> sdp = bootstrap.SDP(0.52, table2)
>>> result1 = sdp.opemax(3.0, 2)
>>> float(result1)
18.76458372724582
```

Here, we have chosen to maximize the coefficient of a spin-2 operator evaluated $\Delta_{\text{unitary}} = 3$. In other words, this bounds λ_T^2, the coefficient of the stress-energy tensor. The returned value is careful to account for the positive prefactors $\chi_\ell(\Delta)$ but necessarily still depends on our normalization convention for conformal blocks. The best way to extract the physical information is to compare this to another OPE coefficient. Below, we do this at the

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

(almost) free field theory point

```
>>> sdp = bootstrap.SDP(0.5001, table2)
>>> result2 = sdp.opemax(3.0, 2)
>>> float(result2)
16.14053403869564
```

Dividing one result by the other causes any extra factors to cancel out. Because we are dealing with the stress-energy tensor, we have enough information to find the central charge

$$C_T = \frac{d}{d-1}\left(\frac{\Delta_\phi}{\lambda_T}\right)^2. \tag{4.1}$$

Checking that the one from `result1` is about 93% of the one from `result2`, we have verified the results of [9, 11] which studied the central charge in the vicinity of the Ising point.

4.6 Global symmetry

The examples so far have all treated a single crossing equation. One way to go beyond this is to give our external scalars flavour indices under some global Lie group symmetry. Fundamentals of $SO(N)$ provide a simple yet important example. Their OPE may be written schematically as

$$\phi_i \times \phi_j \sim \sum_{\mathcal{O}\in S} \delta_{ij}\mathcal{O} + \sum_{\mathcal{O}\in T} \mathcal{O}_{\{i,j\}} + \sum_{\mathcal{O}\in A} \mathcal{O}_{[i,j]}, \tag{4.2}$$

in terms of singlet, traceless symmetric and antisymmetric tensor structures. Terms with Fermi symmetry may only couple to odd spins just as Bose symmetry allowed us to only consider even spins before. Keeping all spins in the tables we prepare, we also need to perform symmetric and antisymmetric convolutions

```
>>> from bootstrap import *
>>> table1 = ConformalBlockTable(3, 15, 15, 1, 3, odd_spins = True)
>>> table2 = ConvolvedBlockTable(table1, symmetric = True)
>>> table3 = ConvolvedBlockTable(table1)
```

Now that our tables include odd spins, our sign convention for conformal blocks becomes especially important. Being careful with this, the sum rule that follows from (4.2) is

$$\sum_{\mathcal{O}\in S} \lambda_{\mathcal{O}}^2 \begin{bmatrix} 0 \\ F_{-,\Delta,\ell} \\ F_{+,\Delta,\ell} \end{bmatrix} + \sum_{\mathcal{O}\in T} \lambda_{\mathcal{O}}^2 \begin{bmatrix} F_{-,\Delta,\ell} \\ \left(1-\frac{2}{N}\right)F_{-,\Delta,\ell} \\ -\left(1+\frac{2}{N}\right)F_{+,\Delta,\ell} \end{bmatrix} + \sum_{\mathcal{O}\in A} \lambda_{\mathcal{O}}^2 \begin{bmatrix} F_{-,\Delta,\ell} \\ -F_{-,\Delta,\ell} \\ F_{+,\Delta,\ell} \end{bmatrix} = 0. \tag{4.3}$$

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

The last vector differs from what appears in [10] by a sign. The presence of $(-1)^\ell$ in (2.4) is what tells us to introduce this sign when using `PyCFTBoot`. It can be seen in [26] that the same authors have now switched to the normalization used here. Because the dimensions of the ϕ_i are all the same, entries of these vectors may be specified with two numbers instead of four. It is easy to do this after choosing an N and a list of tables

```
>>> N = 3.0
>>> table_list = [table2, table3]
>>> vec1 = [[0, 1], [1, 1], [1, 0]]
>>> vec2 = [[1, 1], [1.0 - (2.0 / N), 1], [-(1.0 + (2.0 / N)), 0]]
>>> vec3 = [[1, 1], [-1, 1], [1, 0]]
```

To formulate (4.3) we just need to give these vectors (even, even, odd) spins and (0, 1, 2) representation labels.

```
>>> info = [[vec1, 0, 0], [vec2, 0, 1], [vec3, 1, 2]]
>>> sdp = SDP(0.52, table_list, vector_types = info)
>>> result = sdp.bisect(0.7, 1.8, 0.01, [0, 0])
>>> result
1.6453125000000002
```

This value is approximately what it should be, looking at the bound on singlet scalars produced in [10].

4.7 Mixed correlators

Applying the above methods when four point functions have arbitrary scaling dimensions represents an important advance for the bootstrap. This can reveal a wealth of information even for the simplest CFTs because OPEs with no dimension difference are blind to $\mathbb{Z}_2$-odd operators. Following [24] where many more details can be found, we present an example with only $\mathbb{Z}_2$ symmetry. The even $\epsilon \in E$ is now more than just an internal operator being summed over. It is a member of the four point function on the same level as the odd $\sigma \in O$. Letting the indices in (3.3) run over all combinations of σ and ϵ, we derive a number of crossing equations that can be put into matrix form.

$$\sum_{\mathcal{O}\in E,2|\ell} (\lambda_{\sigma\sigma\mathcal{O}}\lambda_{\epsilon\epsilon\mathcal{O}}) V_{E,\Delta,\ell} \begin{pmatrix} \lambda_{\sigma\sigma\mathcal{O}} \\ \lambda_{\epsilon\epsilon\mathcal{O}} \end{pmatrix} + \sum_{\mathcal{O}\in O,2|\ell} \lambda^2_{\sigma\epsilon\mathcal{O}} V_{O+,\Delta,\ell} + \sum_{\mathcal{O}\in O,2\nmid\ell} \lambda^2_{\sigma\epsilon\mathcal{O}} V_{O-,\Delta,\ell} = 0, \quad (4.4)$$

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

where

$$V_{E,\Delta,\ell}=\begin{bmatrix} \begin{pmatrix} F^{\sigma\sigma;\sigma\sigma}_{-,\Delta,\ell} & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & F^{\epsilon\epsilon;\epsilon\epsilon}_{-,\Delta,\ell} \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & \frac{1}{2}F^{\sigma\sigma;\epsilon\epsilon}_{-,\Delta,\ell} \\ \frac{1}{2}F^{\sigma\sigma;\epsilon\epsilon}_{-,\Delta,\ell} & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & \frac{1}{2}F^{\sigma\sigma;\epsilon\epsilon}_{+,\Delta,\ell} \\ \frac{1}{2}F^{\sigma\sigma;\epsilon\epsilon}_{+,\Delta,\ell} & 0 \end{pmatrix} \end{bmatrix}, \quad V_{O+,\Delta,\ell}=\begin{bmatrix} 0 \\ 0 \\ F^{\sigma\epsilon;\sigma\epsilon}_{-,\Delta,\ell} \\ F^{\epsilon\sigma;\sigma\epsilon}_{-,\Delta,\ell} \\ -F^{\epsilon\sigma;\sigma\epsilon}_{+,\Delta,\ell} \end{bmatrix}, \quad V_{O-,\Delta,\ell}=\begin{bmatrix} 0 \\ 0 \\ F^{\sigma\epsilon;\sigma\epsilon}_{-,\Delta,\ell} \\ -F^{\epsilon\sigma;\sigma\epsilon}_{-,\Delta,\ell} \\ F^{\epsilon\sigma;\sigma\epsilon}_{+,\Delta,\ell} \end{bmatrix}. \tag{4.5}$$

The only difference with respect to [24] is our choice not to use factors of $(-1)^\ell$ to combine the O sums over even and odd spins in (4.4). Looking at all possible dimension differences, there are three conformal block tables to make which give rise to five convolutions.

```
>>> from bootstrap import *
>>> sig = 0.518
>>> eps = 1.412
>>> g_tab1 = ConformalBlockTable(3, 20, 20, 2, 4)
>>> g_tab2 = ConformalBlockTable(3, 20, 20, 2, 4, \
...  eps - sig, sig - eps, odd_spins = True)
>>> g_tab3 = ConformalBlockTable(3, 20, 20, 2, 4, \
...  sig - eps, sig - eps, odd_spins = True)
>>> f_tab1a = ConvolvedBlockTable(g_tab1)
>>> f_tab1s = ConvolvedBlockTable(g_tab1, symmetric = True)
>>> f_tab2a = ConvolvedBlockTable(g_tab2)
>>> f_tab2s = ConvolvedBlockTable(g_tab2, symmetric = True)
>>> f_tab3 = ConvolvedBlockTable(g_tab3)
>>> dim_list = [sig, eps]
>>> tab_list = [f_tab1a, f_tab1s, f_tab2a, f_tab2s, f_tab3]
```

Entering $V_{O\pm,\Delta,\ell}$ is similar to our syntax for the vectors in (4.3). However, we need two `dim_list` indices after the `tab_list` index to specify the inner σs and ϵs

```
>>> v2 = [[0, 0, 0, 0], [0, 0, 0, 0], [1, 4, 1, 0], [1, 2, 0, 0], [-1, 3, 0, 0]]
>>> v3 = [[0, 0, 0, 0], [0, 0, 0, 0], [1, 4, 1, 0], [-1, 2, 0, 0], [1, 3, 0, 0]]
```

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

To continue with $V_{E,\Delta,\ell}$, each entry should be a 2×2 matrix in the standard Python notation

```
>>> m1 = [[[1, 0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0, 0], [0, 0, 0, 0]]]
>>> m2 = [[[0, 0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0, 0], [1, 0, 1, 1]]]
>>> m3 = [[[0, 0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0, 0], [0, 0, 0, 0]]]
>>> m4 = [[[0, 0, 0, 0], [0.5, 0, 0, 1]], [[0.5, 0, 0, 1], [0, 0, 0, 0]]]
>>> m5 = [[[0, 1, 0, 0], [0.5, 1, 0, 1]], [[0.5, 1, 0, 1], [0, 1, 0, 0]]]
>>> v1 = [m1, m2, m3, m4, m5]
```

Folding these into the final argument of `SDP` and iterating can now be done in a familiar way. We should also use an option to ensure that anything taking much longer than a primal feasible problem is correctly recognized as a dual feasible problem

```
>>> info = [[v1, 0, 0], [v2, 0, 1], [v3, 1, 2]]
>>> sdp = SDP(dim_list, tab_list, vector_types = info)
>>> sdp.set_option("dualErrorThreshold", 1e-15)
>>> sdp.add_point([0, 2], sig)
>>> sdp.set_bound([0, 2], 3.0)
>>> sdp.set_bound([0, 0], eps)
>>> result = sdp.iterate()
>>> result
True
```

The only $\mathbb{Z}_2$-even bound we have set is the one that defines ϵ. Instead, the power of this bound comes from our $\mathbb{Z}_2$-odd statement, that every such operator except σ has $\Delta > 3$. Repeating this example with `(sig, eps) = (0.518, 1.2)` returns `False` which is exactly what we saw with the $\mathbb{Z}_2$-even gap before. However, this time we can also rule out CFTs by going "right" of the Ising model in $(\Delta_\sigma, \Delta_\epsilon)$ space. Since `(sig, eps) = (0.53, 1.412)` also returns `False`, we begin to see hints that we are exploring an isolated region of allowed scaling dimensions.

4.8 A superconformal example

Among the known constructions of conformal field theories, examples without supersymmetry are relatively rare. In adapting our CFT bootstrap methods to handle SCFTs, the main new step is combining conformal blocks for different operators in the same multiplet. For the example of 4D $\mathcal{N}=1$ chiral primaries, [17] found the following blocks

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

adding to the results of [61]:

$$\mathcal{G}_{\Delta,\ell}=g_{\Delta,\ell}-\frac{(\ell+2)(\Delta+\ell)}{(\ell+1)(\Delta+\ell+1)}g_{\Delta+1,\ell+1}-\frac{\ell(\Delta-\ell-2)}{(\ell+1)(\Delta-\ell-1)}g_{\Delta+1,\ell-1}+\frac{(\Delta+\ell)(\Delta-\ell-2)}{(\Delta+\ell+1)(\Delta-\ell-1)}g_{\Delta+2,\ell}. \tag{4.6}$$

The R-symmetry in this case is $U(1)\simeq SO(2)$ which allows us to write

$$\sum_{\mathcal{O}\in S,2|\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}F_{-,\Delta,\ell}\\F_{-,\Delta,\ell}\\F_{+,\Delta,\ell}\end{bmatrix}+\sum_{\mathcal{O}\in T,2|\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}0\\2F_{-,\Delta,\ell}\\-2F_{+,\Delta,\ell}\end{bmatrix}+\sum_{\mathcal{O}\in S,2\nmid\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}-F_{-,\Delta,\ell}\\F_{-,\Delta,\ell}\\F_{+,\Delta,\ell}\end{bmatrix}=0. \tag{4.7}$$

One change compared to (4.3) is that we have recognized antisymmetric A operators as simply being odd-spin singlets. The other is that we have replaced the middle row with itself plus twice the top row and replaced the top row with the middle row. Due to the $(-1)^\ell$ factor in (2.4), we have paid attention to the middle two terms of (4.6) and the last term of (4.7). Otherwise the normalization of our $\mathcal{N}=1$ block is the same as that of [27]. Seeing the difference between the first two rows above, it becomes clear that a final sum rule will also have to involve

$$\tilde{\mathcal{G}}_{\Delta,\ell}=g_{\Delta,\ell}+\frac{(\ell+2)(\Delta+\ell)}{(\ell+1)(\Delta+\ell+1)}g_{\Delta+1,\ell+1}+\frac{\ell(\Delta-\ell-2)}{(\ell+1)(\Delta-\ell-1)}g_{\Delta+1,\ell-1}+\frac{(\Delta+\ell)(\Delta-\ell-2)}{(\Delta+\ell+1)(\Delta-\ell-1)}g_{\Delta+2,\ell}. \tag{4.8}$$

Lines that encode this in `PyCFTBoot` are

```
>>> import bootstrap
>>> c1 = (delta + ell + 1) * (delta - ell - 1) * (ell + 1)
>>> c2 = -(delta + ell) * (delta - ell - 1) * (ell + 2)
>>> c3 = -(delta - ell - 2) * (delta + ell + 1) * ell
>>> c4 = (delta + ell) * (delta - ell - 2) * (ell + 1)
>>> combo1 = [[c1, 0, 0], [c2, 1, 1], [c3, 1, -1], [c4, 2, 0]]
>>> combo2 = combo1
>>> combo2[1][0] *= -1
>>> combo2[2][0] *= -1
```

where each triple has a coefficient, a shift in Δ and then a shift in ℓ. Uncharged operators in S come from OPEs of the form $\Phi\times\Phi^\dagger$. These are the ones that have three other operators related by supersymmetry. There are also the T operators from $\Phi\times\Phi$ OPEs which only make use of regular conformal blocks. Allocating all of the tables we need,

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

it is advisable to keep many derivatives because the convergence of 4D $\mathcal{N}=1$ bounds is notoriously slow

```
>>> g_tab = ConformalBlockTable(3.99, 25, 26, 3, 5, odd_spins = True)
>>> f_tab1a = ConvolvedBlockTable(g_tab)
>>> f_tab1s = ConvolvedBlockTable(g_tab, symmetric = True)
>>> f_tab2a = ConvolvedBlockTable(g_tab, content = combo1)
>>> f_tab2s = ConvolvedBlockTable(g_tab, symmetric = True, content = combo1)
>>> f_tab3 = ConvolvedBlockTable(g_tab, content = combo2)
>>> tab_list = [f_tab1a, f_tab1s, f_tab2a, f_tab2s, f_tab3]
```

With 26 spins kept above, the spins of our singlet operators will go up to 25 because each superconformal block draws from the spin above it. The normalization in (4.6) is convenient because the spin-0 expression gives a vanishing coefficient to its $\ell-1$ term. If this were not the case, `PyCFTBoot` would still skip any terms telling us to naively include negative spin. The main remaining task is to write (4.7) in terms of superconformal blocks and absorb a factor of 2 for convenience

$$\sum_{\mathcal{O}\in S,2|\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}\tilde{\mathcal{F}}_{-,\Delta,\ell}\\ \mathcal{F}_{-,\Delta,\ell}\\ \mathcal{F}_{+,\Delta,\ell}\end{bmatrix}+\sum_{\mathcal{O}\in S,2\nmid\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}-\tilde{\mathcal{F}}_{-,\Delta,\ell}\\ \mathcal{F}_{-,\Delta,\ell}\\ \mathcal{F}_{+,\Delta,\ell}\end{bmatrix}+\sum_{\mathcal{O}\in T,2|\ell}\lambda_{\mathcal{O}}^2\begin{bmatrix}0\\ F_{-,\Delta,\ell}\\ -F_{+,\Delta,\ell}\end{bmatrix}=0. \tag{4.9}$$

We will now enter this as an `SDP` at an external dimension of $\Delta_\phi=1.4$.

```
>>> vec1 = [[1, 4], [1, 2], [1, 3]]
>>> vec2 = [[-1, 4], [1, 2], [1, 3]]
>>> vec3 = [[0, 0], [1, 0], [-1, 1]]
>>> info = [[vec1, 0, 0], [vec2, 1, 1], [vec3, 0, 2]]
>>> sdp = SDP(1.4, tab_list, vector_types = info)
```

Since T is constrained by more than just unitarity, we need to set the bound $\Delta_{\text{min}}=|2\Delta_\phi-3|+3+\ell$. The only dimensions lower than this in the charged sector belong to BPS operators with $\Delta=2\Delta_\phi+\ell$. Finishing the computation of this bound,

```
>>> sdp.set_option("dualErrorThreshold", 1e-22)
>>> for l in range(0, 27, 2):
...   sdp.set_bound([l, 2], abs(2 * 1.4 - 3) + 3 + l)
...   sdp.add_point([l, 2], 2 * 1.4 + l)
...
>>> result = sdp.bisect(3.0, 6.0, 0.01, [0, 0])
>>> result
3.966796875
```

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

This is still about 20% away from the known value where a special theory is conjectured to live. The properties of this kink have been studied extensively in [30].

5 A longer example

As a final demonstration, we use `PyCFTBoot` to investigate operator dimensions on the line of critical theories interpolating between the Ising model in $d=3$ and the free boson in $d=4$. The most standard argument for these theories comes from the perturbative analysis of Wilson and Fisher [62]. Perturbation theory yields insight because ϕ^4 theory, which is infrared free in 4 dimensions, has a weakly coupled IR fixed point when the dimension is analytically continued to $4-\varepsilon$. Since they come from a theory with $\mathbb{Z}_2$ symmetry, all of these fixed points should be in the Ising model's universality class. The operators ϕ and ϕ^2 become what we have been calling σ and ϵ — the scalar of lowest dimension and the first scalar appearing in the simplest OPE. Making only a unitarity assumption, [37] used the bootstrap to go beyond perturbation theory and place upper bounds on Δ_ϵ in terms of Δ_σ over the whole range of dimensions. Our goal is to constrain $(\Delta_\sigma,\Delta_\epsilon)$ space further by using the same assumptions that have been successful with the 3D Ising model [23,24]. Namely, we demand that only a single $\mathbb{Z}_2$-odd scalar has a dimension below d.

Since only this gap and the conformal blocks depend on d, we use the same crossing equations as (4.4). For our truncation parameters, we choose $k_{\max}=30$, $\ell_{\max}=20$, $m_{\max}=3$ and $n_{\max}=5$. The only non-default parameters of `SDPB` are `--precision=660 --dualErrorThreshold=1e-15`. Allowed CFTs return `found primal feasible solution` well before this. Due to the number of points that must be checked, it does not make sense to call `ConformalBlockTable` every time we get to a new point. It is common for two different points in the region being scanned to have the same dimension differences. For this reason, we use `PyCFTBoot` to dump all of the tables beforehand and then setup `SDP`s as a second step. Each `SDP` is allocated with the `prototype = old_sdp` argument since almost all parts of the bilinear basis are shared. The resolutions chosen for our scans are given in Table 5. When a pair of anomalous dimensions is found to be compatible with crossing symmetry, a pixel of the appropriate width and height is drawn. Anomalous dimensions are defined via

$$\gamma_\sigma=\Delta_\sigma-\frac{d-2}{2},\quad \gamma_\epsilon=\Delta_\epsilon-(d-2). \tag{5.1}$$

Table 5: Horizontal and vertical spacing between points checked for primal / dual feasibility.

d	Horizontal step	Vertical step
3	0.0005	0.005
3.25	0.0001	0.001
3.5	0.0001	0.001
3.75	0.00005	0.0005

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

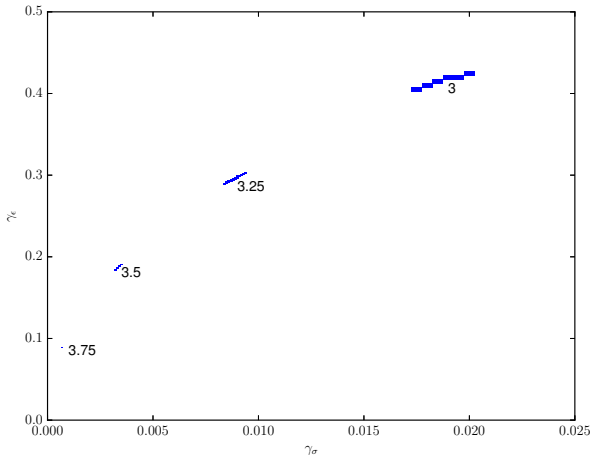


Figure 1: Plot of allowed anomalous dimensions in $d \in \{3,3.25,3.5,3.75\}$. For each d, there is a closed region of points that cannot be excluded using the constraints of the conformal bootstrap on the correlators $\langle\sigma\sigma\sigma\sigma\rangle$, $\langle\sigma\sigma\epsilon\epsilon\rangle$, $\langle\epsilon\epsilon\epsilon\epsilon\rangle$. Their positions have good agreement with the dimensions of the Wilson-Fisher fixed points calculated with the ε-expansion.

Results of these scans are shown in Fig. 1. The well known 3D island occupies the top right corner. In the bottom left corner is a barely visible island for $d = 3.75$. It consists of just three points centred at $(\Delta_\sigma,\Delta_\epsilon) \approx (0.8757,1.839)$. Although it is likely that larger regions of non-excluded points exist away from each island, we have not attempted to search for them.

Numerical checks indicate that the islands shrink as we increase the dimension of our search space. However, their characteristic sizes for a fixed number of derivatives clearly depend on d. Whereas the error bar on Δ_σ from our 3D Ising scan is about 0.005, the same computational resources put toward the Wilson-Fisher fixed point in $d = 3.75$ give us an error bar that is two orders of magnitude smaller. It is in fact comparable in size to the second smallest island found for the 3D Ising model in [23]. This makes sense because increasing d brings us closer to the perturbative regime where error bars from a numerical technique like the bootstrap are not needed at all. This trend is also the reason why we have not decreased d further. Below 3, the islands continue to grow until they merge with the unbounded regions. In the extreme case of $d = 2$, the same plot that follows purely from crossing symmetry and unitarity is returned with the extra assumptions here having no effect. We thank Anton de la Fuente for pointing this out.

6 Discussions

The last example, intended merely to demonstrate the capabilities of `PyCFTBoot`, was done with much less CPU time than bootstrap calculations designed to break precision records. Indeed, this has not been accomplished. Using Borel summation and agreement

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

with 2D values as a boundary condition, [63] has calculated Wilson-Fisher critical exponents up to fifth order in $\varepsilon=4-d$. Their values for $d=3.75$ converted to conformal scaling dimensions are

$$\Delta_\sigma = 0.875718 \pm 0.000005, \quad \Delta_\epsilon = 1.83943 \pm 0.00005. \tag{6.1}$$

Our bootstrap result, which is in complete agreement, does not fix as many digits. Moreover, it is not necessarily true that our error bars, found by rigorously excluding points are safer to use than those found by resummation techniques. The caveat that prevents this interpretation is the tacit assumption that the Wilson-Fisher fixed point is unitary. While this assumption has long been made, it is incorrect according to a recent analysis which takes non-integer d seriously [64]. A striking result is their finding that four descendants in the spectrum have the complex dimensions

$$\begin{aligned}\Delta &= 23+\left(\frac{\lambda}{36}-\frac{7}{2}\right)\varepsilon+\mathcal{O}(\varepsilon^2),\\ &\lambda \in \{16.93372103 \pm 5.59469106i, 42.88540243 \pm 1.07557547i\}.\end{aligned} \tag{6.2}$$

These evanescent operators, as they are called, have correlation functions with more familiar operators like ϕ and ϕ^2 that only vanish when d is an integer. The primaries from which they arise are guaranteed to have $\Delta \geq 15$ [64].

Unlike some more approximate schemes [65], the methods used in `SDPB` and similar codes cannot rule out trial spectra containing complex dimensions. Complex dimensions would motivate us to consider polynomials in $x+iy$ instead of just x. Since dimensions occur in conjugate pairs, the only natural domain for y would be all of $\mathbb{R}$. This is incompatible with the positivity conditions of (3.4) which have to be phrased on a half-line[‡‡]. Even if this problem could be solved, adapting the method to non-unitary theories would also require a way of dealing with complex OPE coefficients. It is important then to discuss why Figure 1 still appears to show four reasonable islands.

For d sufficiently close to 3, it is obvious that an island will still be found. The bootstrap, like any well-posed computational problem, is robust to small changes in the parameters. These parameters include the spatial d and also the dimensionality of Lie group symmetries that are present [16, 33]. On the other hand, there have already been situations where d is fractional enough for the bootstrap to rule out all unitary theories [66]. Parameters that impact our ability to make this distinction are the number of poles and the number of derivatives. There are three possibilities for what happens as they are increased:

1. The islands disappear when these parameters reach certain large but finite numbers.

[‡‡]When a polynomial has odd degree for example, it can never be positive for all $x \in \mathbb{R}$. Neglecting odd degrees does not make sense because a conformal block approximation can never become worse when its degree increases by one.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

2. The islands persist because some undiscovered unitary CFT has a low-lying spectrum very similar to that of the Wilson-Fisher fixed point.

3. The islands persist for some other reason.

We conjecture that the first option is realized. In particular, one's ability to find a fake "Wilson-Fisher island" with arbitrarily high precision would cast doubt on the conventional wisdom for what happens to the Ising model's island in $d=3$. A mixed correlator bootstrap that lacks the power to rule out some crossing asymmetric points in $3<d<4$, would probably also cause the 3D island to converge to a finite size. While this is a possible topic for future work, it is also likely to be a difficult one. The island for $d=3.75$ is only small compared to the scale of Fig. 1. We have no reason to believe that it is anywhere close to disappearing.

Apart from this, there are still many unitary theories that can benefit from a conformal bootstrap treatment. `PyCFTBoot` can allow these studies to happen more quickly and serve as a starting point for those wishing to test modifications to the various algorithms. Adding code to deal with more general conformal blocks is an important next step. Constraints from external tensor and spinor operators are expected to shed light on a number of previously unexplored theories in $2<d<6$. They could also help answer the still open question of whether interacting CFTs above six dimensions can exist.

The list of example problems that `PyCFTBoot` can handle is already fairly large. We expect this to grow as the community makes progress on important phenomenological questions at an increasing rate. If bugs are encountered along the way, anyone can read the code of `PyCFTBoot` or one of its dependencies in order to suggest a fix. A few flagship results of the bootstrap have become widely known and the pool of introductory papers is of course larger than it has ever been. Now that adequate software is available for bootstrapping a CFT from start to finish, the time is ripe to get new people involved.

Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada. The numerics were performed on the Pheno cluster in the C. N. Yang Institute for Theoretical Physics. I thank Chi Ming Hung for maintaining this cluster and installing some of the required libraries. Some discussions related to this work also happened during the Simons Summer Workshop 2015. Participants including Agnese Bissi, Alejandro Castedo, Matthijs Hogervorst, Balt van Rees and Alessandro Vichi gave helpful comments while this code was in development. I am especially grateful to Madalena Lemos for sharing advice on all aspects of conformal bootstrap calculations over the course of a year. In the final stages of this work, I received input from Sheer El-Showk, Anton de la Fuente, Leonardo Rastelli, Slava Rychkov and David Simmons-Duffin. After the first release, Junchen Rong found a misprint and Jaehoon Lee helped to debug a crucial normalization error.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

References

[1] S. Ferrara, A. F. Grillo and R. Gatto, Tensor representations of conformal algebra and conformally covariant operator product expansion, Ann. Phys., 76 (1973), 161–188.

[2] A. M. Polyakov, Non-Hamiltonian approach to conformal quantum field theory, J. Exp. Theor. Phys., 66 (1974), 23–42.

[3] J. Maldacena, The large-N limit of superconformal field theories and supergravity, Int. J. Theor. Phys., 38 (1998), 1113–1133, arXiv:9711.200.

[4] R. Rattazzi, S. Rychkov, E. Tonni and A. Vichi, Bounding scalar operator dimensions in 4D CFT, J. High Energy Phys., 12(31) (2008), arXiv:0807.0004.

[5] S. Rychkov and A. Vichi, Universal constraints on conformal operator dimensions, Phys. Rev. D, 80(4) (2009), arXiv:0905.2211.

[6] F. Caracciolo and S. Rychkov, Rigorous limits on the interaction strength in quantum field theory, Phys. Rev. D, 81(8) (2010), arXiv:0912.2726.

[7] R. Rattazzil, S. Rychkov and A. Vichi, Central charge bounds in 4D conformal field theory, Phys. Rev. D, 83(4) (2011). arXiv:1009.2725.

[8] R. Rattazzil, S. Rychkov and A. Vichi, Bounds in 4D conformal field theories with global symmetry, J. Phys. A, 44(3) (2011), arXiv:1009.5985.

[9] S. El-Showk, M. F. Paulos, D. Poland, S. Rychkov, D. Simmons-Duffin and A. Vichi, Solving the 3D Ising model with the conformal bootstrap, Phys. Rev. D, 86(2) (2012), arXiv:1203.6064.

[10] F. Kos, D. Poland and D. Simmons-Duffin, Bootstrapping the $\mathcal{O}(N)$ vector models, J. High Energy Phys., 6(91) (2014), arXiv:1307.6856.

[11] S. El-Showk, M. F. Paulos, D. Poland, S. Rychkov, D. Simmons-Duffin and A. Vichi, Solving the 3D Ising model with the conformal bootstrap II: c-minimization and precise critical exponents, J. Stat. Phys., 157 (2014), 869–914, arXiv:1403.4545.

[12] Y. Nakayama and T. Ohtsuki, Approaching conformal window of $\mathcal{O}(n)\times\mathcal{O}(m)$ symmetric Landau-Ginzburg models from conformal bootstrap, Phys. Rev. D, 89(12) (2014), arXiv:1404.0489.

[13] Y. Nakayama and T. Ohtsuki, Five dimensional $\mathcal{O}(N)$ symmetric CFTs from conformal bootstrap, Phys. Lett. B, 734 (2014), 193–197, arXiv:1404.5201.

[14] Y. Nakayama and T. Ohtsuki, Bootstrapping phase transitions in QCD and frustrated spin systems, Phys. Rev. D, 91(2) (2015), arXiv:1407.6195.

[15] J. B. Bae and S. J. Rey, Conformal bootstrap approach to $\mathcal{O}(N)$ fixed points in five dimensions, (2014), arXiv:1412.6549.

[16] S. M. Chester, S. S. Pufu and R. Yacoby, Bootstrapping $\mathcal{O}(N)$ vector models in $4<d<6$, Phys. Rev. D, 91(8) (2015), arXiv:1412.7746.

[17] D. Poland and D. Simmons-Duffin, Bounds on 4D conformal and superconformal field theories, J. High Energy Phys., 5(17) (2010), arXiv:1009.2087.

[18] D. Poland, D. Simmons-Duffin and A. Vichi, Carving out the space of 4D CFTs, J. High Energy Phys., 5(110) (2012), arXiv:1109.5176.

[19] C. Beem, L. Rastelli and B. C. van Rees, The $\mathcal{N}=4$ superconformal bootstrap, Phys. Rev. Lett., 111 (2013), 071601, arXiv:1304.1803.

[20] S. M. Chester, J. Lee, S. S. Pufu and R. Yacoby, The $\mathcal{N}=8$ superconformal bootstrap in three dimensions, J. High Energy Phys., 9(143) (2014), arXiv:1406.4814.

[21] C. Beem, M. Lemos, P. Liendo, L. Rastelli and B. C. van Rees, The $\mathcal{N}=2$ superconformal bootstrap, J. High Energy Phys., 3(183) (2016), arXiv:1412.7541.

[22] M. F. Paulos, JuliBootS: A hands-on guide to the conformal bootstrap, (2014),

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

arXiv:1412.4127.
[23] D. Simmons-Duffin, A semidefinite program solver for the conformal bootstrap, J. High Energy Phys., 6(174) (2015), arXiv:1502.02033.
[24] F. Kos, D. Poland and D. Simmons-Duffin, Bootstrapping mixed correlators in the 3D Ising model, J. High Energy Phys., 11(109) (2014), arXiv:1406.4858.
[25] D. Swinarski, Software for computing conformal block divisors on $\overline{M}_{0,n}$, (2014), `http://faculty.fordham.edu/dswinarski/conformalblocks/ConformalBlocksAug2014.pdf`.
[26] F. Kos, D. Poland, D. Simmons-Duffin and A. Vichi, Bootstrapping the $\mathcal{O}(N)$ archipelago, J. High Energy Phys., 11(106) (2015), arXiv:1504.07997.
[27] S. M. Chester, S. Giombi, L. V. Iliesiu, I. R. Klebanov, S. S. Pufu and R. Yacoby, Accidental symmetries and the conformal bootstrap, J. High Energy Phys., 1(110) (2016), arXiv:1507.04424.
[28] C. Beem, M. Lemos, L. Rastelli and B. C. van Rees, The $(2,0)$ superconformal bootstrap, Phys. Rev. D, 93(2) (2016), arXiv:1507.05637.
[29] L. V. Iliesiu, F. Kos, D. Poland, S. S. Pufu, D. Simmons-Duffin and R. Yacoby, Bootstrapping 3D fermions, J. High Energy Phys., 3(120) (2016), arXiv:1508.00012.
[30] D. Poland and A. Stergiou, Exploring the minimal 4D $\mathcal{N}=1$ SCFT, J. High Energy Phys., 12(121) (2015), arXiv:1509.06368.
[31] M. Lemos and P. Liendo, Bootstrapping $\mathcal{N}=2$ chiral correlators, J. High Energy Phys., 1(25) (2016), arXiv:1510.03866.
[32] Y. Lin, S. Shao, D. Simmons-Duffin, Y. Wang and X. Yin, $\mathcal{N}=4$ superconformal bootstrap of the K3 CFT, (2015), arXiv:1511.04065.
[33] S. M. Chester, L. V. Iliesiu, S. S. Pufu and R. Yacoby, Bootstrapping $\mathcal{O}(N)$ vector models with four supercharges in $3\leq d\leq 4$, J. High Energy Phys., 5(103) (2016), arXiv:1511.07552.
[34] S. M. Chester and S. S. Pufu, Towards bootstrapping QED3, J. High Energy Phys., 8(19) (2016), arXiv:1601.03476.
[35] S. van der Walt, S. C. Colbert and G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Comput. Sci. Eng., 13 (2011), 22–30.
[36] SymPy Development Team, SymPy: Python Library for Symbolic Mathematics, 2014, `http://www.sympy.org`.
[37] S. El-Showk, M. F. Paulos, D. Poland, S. Rychkov, D. Simmons-Duffin and A. Vichi, Conformal field theories in fractional dimensions, Phys. Rev. Lett., 112 (2014), 141601, arXiv:1309.5089.
[38] M. S. Costa, J. Penedones, D. Poland and S. Rychkov, Spinning conformal correlators, J. High Energy Phys., 11(71) (2011), arXiv:1107.3554.
[39] M. S. Costa, J. Penedones, D. Poland and S. Rychkov, Spinning conformal blocks, J. High Energy Phys., 11(154) (2011), arXiv:1109.6321.
[40] D. Simmons-Duffin, Projectors, shadows and conformal blocks, J. High Energy Phys., 4(146) (2014), arXiv:1204.3894.
[41] W. Siegel, Embedding vs 6D twistors, (2012), arXiv:1204.5679.
[42] H. Osborn, Conformal blocks for arbitrary spins in two dimensions, Phys. Lett. B, 718 (2012), 169–172, arXiv:1205.1941.
[43] A. Dymarsky, On the four-point function of the stress-energy tensors in a CFT, J. High Energy Phys., 10(75) (2015), arXiv:1311.4546.
[44] M. S. Costa and T. Hansen, Conformal correlators of mixed-symmetry tensors, J. High Energy Phys., 2(151) (2015), arXiv:1411.7351.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press

[45] E. Elkhidir; D. Karateev and M. Serone, General three-point functions in 4D CFT, J. High Energy Phys., 1(133) (2015), arXiv:1412.1796.
[46] A. C. Echeverri, E. Elkhidir, D. Karateev and M. Serone, Deconstructing conformal blocks in 4D CFT, J. High Energy Phys., 8(101) (2015), arXiv:1505.03750.
[47] F. Rejon-Barrera and D. Robbins, Scalar-vector bootstrap, J. High Energy Phys., 1(139) (2016), arXiv:1508.02676.
[48] L. V. Iliesiu, F. Kos, D. Poland, S. S. Pufu, D. Simmons-Duffins and R. Yacoby, Fermion-scalar conformal blocks, J. High Energy Phys., 4(74) (2016), arXiv:1511.01497.
[49] A. C. Echeverri, E. Elkhidir, D. Karateev and M. Serone, Seed conformal blocks in 4D CFT, J. High Energy Phys., 2(183) (2016), arXiv:1601.05325.
[50] F. A. Dolan and H. Osborn, Conformal four point functions and the operator product expansion, Nuclear Phys. B, 599 (2001), 459–496, arXiv:hep-th/0011040.
[51] F. A. Dolan and H. Osborn, Conformal partial waves and the operator product expansion, Nuclear Phys. B, 678 (2004), 491–507, arXiv:hep-th/0309180.
[52] A. L. Fitzpatrick, J. Kaplan and D. Poland, Conformal blocks in the large D limit, J. High Energy Phys., 8(107) (2013), arXiv:1305.0004.
[53] M. Hogervorst and S. Rychkov, Radial coordinates for conformal blocks, Phys. Rev. D, 87(10) (2013), arXiv:1303.1111.
[54] J. Penedones, E. Trevisani and M. Yamazaki, Recursion relations for conformal blocks, J. High Energy Phys., 9(70) (2016), arXiv:1509.00428.
[55] R. Dijkgraaf, J. Maldacena, G. Moore and E. Verlinde, A black hole Farey tail, (2000), arXiv:hep-th/0005003.
[56] M. Hogervorst, H. Osborn and S. Rychkov, Diagonal limit for conformal blocks in d dimensions, J. High Energy Phys., 8(14) (2013), arXiv:1305.1321.
[57] S. El-Showk and M. F. Paulos, Bootstrapping conformal field theories with the extremal functional method, Phys. Rev. Lett., 111 (2013), 241601, arXiv:1211.2810.
[58] F. A. Dolan and H. Osborn, Conformal Partial Waves: Further Mathematical Results, 2011, arXiv:1108.6194.
[59] E. E. Tyrtyshnikov, A Brief Introduction to Numerical Analysis, Springer, 2012.
[60] W. Gautschi, ORTHPOL–A package of routines for generating orthogonal polynomials and Gauss-type quadrature rules, ACM Transactions on Mathematical Software, 20 (1994), 21–62, arXiv:math/9307212.
[61] F. A. Dolan and H. Osborn, Superconformal symmetry, correlation functions and the operator product expansion, Nuclear Phys. B, 629 (2002), 3–73, arXiv:hep-th/0112251.
[62] K. G. Wilson and M. E. Fisher, Critical exponents in 3.99 dimensions, Phys. Rev. Lett., 28(4) (1972), 240–243.
[63] J. C. Le Guillo and J. Zinn-Justin, Accurate critical exponents for Ising like systems in non-integer dimensions, Journal de Physique, 48 (1987), 19–24.
[64] M. Hogervorst, S. Rychkov and B. C. van Rees, Univarity violation at Wilson-Fisher fixed point in 4-epsilon dimensions, Phys. Rev. D, 93(12) (2016), arXiv:1512.00013.
[65] F. Gliozzi, Constraints on conformal field theories in diverse dimensions from the bootstrap mechanism, Phys. Rev. Lett., 111 (2013), 161602, arXiv:1307.3111.
[66] J. Golden and M. F. Paulos, No unitary bootstrap for the fractal Ising model, J. High Energy Phys., 3(167) (2015), arXiv:1411.7932.

https://doi.org/10.4208/cicp.OA-2016-0107 Published online by Cambridge University Press