Better tools for crossing equations

Connor Behan

2018-07-16

Scope

- CFT and SCFT in $\mathbb{R}^d$ for (integer) $d \geq 2$.
- Any collection of four-point functions.
- Blocks $G_{\mathcal{O}}^{12;34}(z, \bar{z})$ expanded around $z = \bar{z} = \frac{1}{2}$.

Scope

- CFT and SCFT in $\mathbb{R}^d$ for (integer) $d \geq 2$.
- Any collection of four-point functions.
- Blocks $G_{\mathcal{O}}^{12;34}(z,\bar{z})$ expanded around $z = \bar{z} = \frac{1}{2}$.

Table generator
$\downarrow$
Bridge
$\downarrow$
SDPB

Scope

- CFT and SCFT in $\mathbb{R}^d$ for (integer) $d \geq 2$.
- Any collection of four-point functions.
- Blocks $G_{\mathcal{O}}^{12;34}(z, \bar{z})$ expanded around $z = \bar{z} = \frac{1}{2}$.

Table generator — Project 2 gives us a block table as input

$\downarrow$

Bridge

$\downarrow$

SDPB — Compare output to XML files in Project 1

Existing software

First (and last?) program to try all three was JuliBoots [Paulos, 2014] .

PyCFTBoot [CB, 2016]

- Backend is Symengine written in C++
- Small library, started recently

Existing software

First (and last?) program to try all three was JuliBoots [Paulos, 2014] .

PyCFTBoot [CB, 2016]

- Backend is Symengine written in C++
- Small library, started recently

```
F = ConvolvedBlockTable(G)
dims = [sig, eps]
tabs = [F, ...]
matrices = []
matrices.append([[[0, 0, 0, 0], [0.5, 0, 0, 1]],
                 [[0.5, 0, 0, 1], [0, 0, 0, 0]]])
...
sdp = SDP(dims, tabs, [[matrices, 0, 0], ...])
sdp.write_xml(obj, norm, "island")
```

Existing software

First (and last?) program to try all three was JuliBoots [Paulos, 2014] .

PyCFTBoot [CB, 2016]

- Backend is Symengine written in C++
- Small library, started recently

```
F = ConvolvedBlockTable(G)
dims = [sig, eps]
tabs = [F, ...]
matrices = []
matrices.append([[[0, 0, 0, 0], [0.5, 0, 0, 1]],
                 [[0.5, 0, 0, 1], [0, 0, 0, 0]]])
...
sdp = SDP(dims, tabs, [[matrices, 0, 0], ...])
sdp.write_xml(obj, norm, "island")
```

CBoot [Ohtsuki, 2016]

- Backend is Sage written in everything
- Huge package, more stable

Existing software

First (and last?) program to try all three was JuliBoots [Paulos, 2014] .

PyCFTBoot [CB, 2016]

- Backend is Symengine written in C++
- Small library, started recently

```
F = ConvolvedBlockTable(G)
dims = [sig, eps]
tabs = [F, ...]
matrices = []
matrices.append([[[0, 0, 0, 0], [0.5, 0, 0, 1]],
                 [[0.5, 0, 0, 1], [0, 0, 0, 0]]])
...
sdp = SDP(dims, tabs, [[matrices, 0, 0], ...])
sdp.write_xml(obj, norm, "island")
```

CBoot [Ohtsuki, 2016]

- Backend is Sage written in everything
- Huge package, more stable

```
pvms = []
kernel = context.F_minus_matrix((sig + eps) / 2.0)
for l in spins:
    F = context.dot(kernel, G[l])
    matrices = []
    matrices.append([[0, 0.5 * F],
                     [0.5 * F, 0]])
    ...
    pvms.append(matrices)
sdp = context.sumrule_to_SDP(norm, obj, pvms)
sdp.write("island.xml")
```

Problems

No spin yet (obvious).

Problems

No spin yet (obvious).

Prefactors are stored once for each matrix element.

$$\begin{bmatrix} 0 & \frac{1}{2}\chi_\ell(\Delta)P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) \\ \frac{1}{2}\chi_\ell(\Delta)P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) & 0 \end{bmatrix}$$

$$= \chi_\ell(\Delta)\begin{bmatrix} 0 & \frac{1}{2}P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) \\ \frac{1}{2}P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) & 0 \end{bmatrix}$$

This becomes tricky in larger problems where there are mixed and unmixed blocks in the same matrix.

Problems

No spin yet (obvious).

Prefactors are stored once for each matrix element.

$$\begin{bmatrix} 0 & \frac{1}{2}\chi_\ell(\Delta)P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) \\ \frac{1}{2}\chi_\ell(\Delta)P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) & 0 \end{bmatrix}$$
$$= \chi_\ell(\Delta)\begin{bmatrix} 0 & \frac{1}{2}P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) \\ \frac{1}{2}P^{\sigma\sigma;\epsilon\epsilon}_{-,\ell}(\Delta) & 0 \end{bmatrix}$$

This becomes tricky in larger problems where there are mixed and unmixed blocks in the same matrix.

Many labels are needed. The same row / column might involve $F^{\sigma\epsilon;\sigma\epsilon}_{-,\Delta,\ell}$, $F^{\epsilon\sigma;\sigma\epsilon}_{-,\Delta,\ell}$, $F^{\epsilon\sigma;\sigma\epsilon}_{+,\Delta,\ell}$, etc.

Problems

1. Calculations happen in one process.
2. Loops over spin assume $\ell \in \{0, 1, \ldots, \ell_{\max}\}$ or $\ell \in \{0, 2, \ldots, \ell_{\max}\}$. A high spin like 1000 might be desired.
3. Libraries use a lot of memory.

Problems

1. Calculations happen in one process.
2. Loops over spin assume $\ell \in \{0, 1, \ldots, \ell_{\max}\}$ or $\ell \in \{0, 2, \ldots, \ell_{\max}\}$. A high spin like 1000 might be desired.
3. Libraries use a lot of memory.

Spawn a separate process for each spin and let the user specify how many may run at a time!

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) \quad = \quad \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \bigg|_{\text{crossing}}$$

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\text{crossing}}$$

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial a^m \partial b^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\text{crossing}} : \; a = z + \bar{z}, b = (z - \bar{z})^2$$

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \left. \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \right|_{\text{crossing}}$$

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \left. \frac{\partial^{m+n}}{\partial a^m \partial b^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \right|_{\text{crossing}} : \; a = z + \bar{z}, b = (z - \bar{z})^2$$

Cutoff with $m + n \leq \Lambda$ vs $n \leq n_{\max}, m \leq 2(n_{\max} - n) + m_{\max}$.

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \left. \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \right|_{\text{crossing}}$$

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \left. \frac{\partial^{m+n}}{\partial a^m \partial b^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \right|_{\text{crossing}} : \; a = z + \bar{z}, b = (z - \bar{z})^2$$

Cutoff with $m + n \leq \Lambda$ vs $n \leq n_{\max}, m \leq 2(n_{\max} - n) + m_{\max}$.

```
self.dim = 4
self.k_max = 3
self.l_max = 10
self.m_max = 1
self.n_max = 1
self.delta_12 = 0
self.delta_34 = 0
self.odd_spins = False
self.m_order = [0, 0, 0, 0, 1, 1]
self.n_order = [0, 1, 2, 3, 0, 1]
self.table = []
```

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\text{crossing}}$$

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial a^m \partial b^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\text{crossing}} \; : \; a = z + \bar{z}, b = (z - \bar{z})^2$$

Cutoff with $m + n \leq \Lambda$ vs $n \leq n_{\max}, m \leq 2(n_{\max} - n) + m_{\max}$.

```
self.dim = 4
self.k_max = 3
self.l_max = 10
self.m_max = 1
self.n_max = 1
self.delta_12 = 0
self.delta_34 = 0
self.odd_spins = False
self.m_order = [0, 0, 0, 0, 1, 1]
self.n_order = [0, 1, 2, 3, 0, 1]
self.table = []
```

```
derivatives = []
derivatives.append(1 + 2*delta + 3*delta**2 + 4*delta**3)
derivatives.append(2 + 3*delta + 4*delta**2 + 5*delta**3)
derivatives.append(3 + 4*delta + 5*delta**2 + 6*delta**3)
derivatives.append(4 + 5*delta + 6*delta**2 + 7*delta**3)
derivatives.append(5 + 6*delta + 7*delta**2 + 8*delta**3)
derivatives.append(6 + 7*delta + 8*delta**2 + 9*delta**3)
self.table.append(PolynomialVector(derivatives, [0, 0], [1.0, 2.0, 3.0]))
# Same for spin 2
self.table.append(PolynomialVector(derivatives, [0, 0], [4.0, 5.0, 6.0]))
# Same for 4, 6, 8, 10
```

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\mathrm{crossing}}$$

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial a^m \partial b^n} G_{\Delta,\ell}^{\Delta_{12},\Delta_{34}} \Big|_{\mathrm{crossing}} : \ a = z + \bar{z}, b = (z - \bar{z})^2$$

Cutoff with $m + n \leq \Lambda$ vs $n \leq n_{\mathrm{max}}, m \leq 2(n_{\mathrm{max}} - n) + m_{\mathrm{max}}$.

```
self.dim = 4
self.k_max = 3
self.l_max = 10
self.m_max = 1
self.n_max = 1
self.delta_12 = 0
self.delta_34 = 0
self.odd_spins = False
self.m_order = [0, 0, 0, 0, 1, 1]
self.n_order = [0, 1, 2, 3, 0, 1]
self.table = []
```

```
derivatives = []
derivatives.append(1 + 2*delta + 3*delta**2 + 4*delta**3)
derivatives.append(2 + 3*delta + 4*delta**2 + 5*delta**3)
derivatives.append(3 + 4*delta + 5*delta**2 + 6*delta**3)
derivatives.append(4 + 5*delta + 6*delta**2 + 7*delta**3)
derivatives.append(5 + 6*delta + 7*delta**2 + 8*delta**3)
derivatives.append(6 + 7*delta + 8*delta**2 + 9*delta**3)
self.table.append(PolynomialVector(derivatives, [0, 0], [1.0, 2.0, 3.0]))
# Same for spin 2
self.table.append(PolynomialVector(derivatives, [0, 0], [4.0, 5.0, 6.0]))
# Same for 4, 6, 8, 10
```

Everyone agrees on what “dim” is...

$k_{\mathrm{max}}, \ell_{\mathrm{max}}, m_{\mathrm{max}}, n_{\mathrm{max}}$ not so much.

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\rho,l}^{\Delta_i,\rho_i,a,b} \bigg|_{\text{crossing}}$$

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\rho,l}^{\Delta_i,\rho_i,a,b}\Big|_{\text{crossing}}$$

```
<dim>4</dim>
<irrep1>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep1>
<irrep2>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep2>
<irrep3>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep3>
<irrep4>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep4>
```

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n}}{\partial z^m \partial \bar{z}^n} G_{\Delta,\rho,l}^{\Delta_i,\rho_i,a,b}\Big|_{\text{crossing}}$$

```
<dim>4</dim>
<irrep1>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep1>
<irrep2>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep2>
<irrep3>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep3>
<irrep4>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep4>
```

```
<block>
    <m1>4</m1>
    <m2>0</m2>

    <3ptLeft>?</3ptLeft>
    <3ptRight>?</3ptRight>
    <4pt>?</4pt>

    <polynomialVector>
        <polynomial>
            <aOrder>0</aOrder>
            <bOrder>0</bOrder>
            <zOrder>1</zOrder>
            <zbOrder>1</zbOrder>
            <coeff>42</coeff>
            <coeff>137</coeff>
        </polynomial>
    </polynomialVector>

    <poles>
        <elt>1.0</elt>
        <elt>2.0</elt>
        <elt>3.0</elt>
    </poles>
</block>
```

```
<block>
    <m1>2</m1>
    <m2>1</m2>

    <3ptLeft>?</3ptLeft>
    <3ptRight>?</3ptRight>
    <4pt>?</4pt>

    <polynomialVector>
        <polynomial>
            <aOrder>0</aOrder>
            <bOrder>0</bOrder>
            <zOrder>1</zOrder>
            <zbOrder>1</zbOrder>
            <coeff>26</coeff>
            <coeff>10</coeff>
        </polynomial>
    </polynomialVector>

    <poles>
        <elt>1.0</elt>
        <elt>2.0</elt>
        <elt>3.0</elt>
    </poles>
</block>
```

Table formats

$$\chi_\ell(\Delta) P_\ell^{mn}(\Delta) = \frac{\partial^{m+n+p+q}}{\partial z^m \partial \bar{z}^n \partial a^p \partial b^q} G^{\Delta_i,\rho_i,a,b}_{\Delta,\rho,l}\Big|_{\text{crossing}}$$

```
<dim>4</dim>
<irrep1>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep1>
<irrep2>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep2>
<irrep3>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep3>
<irrep4>
    <delta>4</delta>
    <m1>2</m1>
    <m2>0</m2>
    <label>0</label>
</irrep4>
```

```
<block>
    <m1>4</m1>
    <m2>0</m2>

    <3ptLeft>?</3ptLeft>
    <3ptRight>?</3ptRight>
    <4pt>?</4pt>

    <polynomialVector>
        <polynomial>
            <aOrder>0</aOrder>
            <bOrder>0</bOrder>
            <zOrder>1</zOrder>
            <zbOrder>1</zbOrder>
            <coeff>42</coeff>
            <coeff>137</coeff>
        </polynomial>
    </polynomialVector>

    <poles>
        <elt>1.0</elt>
        <elt>2.0</elt>
        <elt>3.0</elt>
    </poles>
</block>
```

```
<block>
    <m1>2</m1>
    <m2>1</m2>

    <3ptLeft>?</3ptLeft>
    <3ptRight>?</3ptRight>
    <4pt>?</4pt>

    <polynomialVector>
        <polynomial>
            <aOrder>0</aOrder>
            <bOrder>0</bOrder>
            <zOrder>1</zOrder>
            <zbOrder>1</zbOrder>
            <coeff>26</coeff>
            <coeff>10</coeff>
        </polynomial>
    </polynomialVector>

    <poles>
        <elt>1.0</elt>
        <elt>2.0</elt>
        <elt>3.0</elt>
    </poles>
</block>
```

What language?

Customization is inevitable.

What language?

Customization is inevitable. Consider $20'$ multiplet in 4D $\mathcal{N} = 4$.

$$u^2 G(v,u) - v^2 G(u,v) + 4(v^2 - u^2) + \frac{4}{c}(v-u) = F_{\mathrm{short}}(u,v,c)$$

What language?

Customization is inevitable. Consider $20'$ multiplet in 4D $\mathcal{N} = 4$.

$$u^2 G(v,u) - v^2 G(u,v) + 4(v^2 - u^2) + \frac{4}{c}(v - u) = F_{\rm short}(u,v,c)$$

Need a way of inputting result of [Beem, Rastelli, van Rees, 2013] .

$$G_{\rm short}(z,\bar{z}) = \frac{12\log(1-z)\log(1-\bar{z})}{|z|^4}\left(2 + \frac{3}{c}\right) + \ldots$$

What language?

Customization is inevitable. Consider $20'$ multiplet in 4D $\mathcal{N} = 4$.

$$u^2 G(v, u) - v^2 G(u, v) + 4(v^2 - u^2) + \frac{4}{c}(v - u) = F_{\text{short}}(u, v, c)$$

Need a way of inputting result of [Beem, Rastelli, van Rees, 2013] .

$$G_{\text{short}}(z, \bar{z}) = \frac{12 \log(1 - z) \log(1 - \bar{z})}{|z|^4} \left(2 + \frac{3}{c}\right) + \ldots$$

```
def new_unit(a, b, c):
    z = 0.5 * (a + sqrt(b))
    zb = 0.5 * (a - sqrt(b))
    return (48 + (36 / c)) * log(1 - z) * log(1 - zb) / (z * zb) ** 2 + ...

F = ConvolvedBlockTable(G)
sdp = SDP(2.0, F)

for i in range(0, len(sdp.unit)):
    m = sdp.m_order[i]
    n = sdp.n_order[i]
    deriv = new_unit(a, b, 0.75).diff(a, m).diff(b, n)
    sdp.unit[i] = deriv.subs(a, 1).subs(b, 0)
```

Path forward

- Start a git repository.
- Explore options for parsing XML [Simmons-Duffin, 2016] .
- Find a way to convolve each irrep in a separate process.
- Allow the "dim_ext" argument to be symbolic (better for speed) or numerical (better for memory).
- Give people a way to save blocks again after convolution.
- Copy relevant parts of PyCFTBoot / CBoot.