Notes on BCFW Recursion in MATHEMATICA

Jacob L. Bourjaily

Niels Bohr International Academy and Discovery Center, Niels Bohr Institute
University of Copenhagen, Blegdamsvej 17, DK-2100, Copenhagen Ø, Denmark

ABSTRACT: Basic reference formulary and key functions know for coding BCFW recursion in MATHEMATICA, prepared for the *International School on Amplitudes, Cosmology, Holography, and Positive Geometries* in Lecce, Italy, 2019.

Contents

1   Symbolic BCFW Recursion in Momentum-Twistor Space

We will denote the n-point $\mathrm{N}^k\mathrm{MHV}$ tree-amplitude as '$\mathcal{A}_n^{(k)}$'. When expressed in terms of momentum-twistor variables, MHV ($k\!=\!0$) amplitudes are the identity for all multiplicity, $\mathcal{A}_n^{(0)}\!=\!1$, and amplitudes vanish when $k\!<\!0$ or $k\!>\!(n\!-\!4)$ (for $n\!>\!3$).

The momenta of the particles involved in an amplitude are encoded by momentum (super)twistors denoted 'z_a'. When it is important or necessary to make clear *which* set of momentum twistors are involved in an amplitude, this will be indicated by writing '$\mathcal{A}_n^{(k)}(a,b,\ldots)$' for $\mathcal{A}_n^{(k)}(z_a,z_b,\ldots)$.

The BCFW recursion relations allow us to determine the n-point $\mathrm{N}^k\mathrm{MHV}$ tree-amplitude $\mathcal{A}_n^{(k)}$ in terms of lower-point trees as follows:

$$
\mathcal{A}_n^{(k)}\big(1,\ldots,n\big) = \ \mathcal{A}_{n-1}^{(k)}\big(1,\ldots,n\!-\!1\big) \tag{1.1}
$$
$$
+\sum_{\substack{n_L+n_R=n+2\\k_L+k_R=k-1}}\mathcal{A}_{n_L}^{(k_L)}\big(1,\ldots,a\!-\!1,\widehat{a}\big)R[1,a\!-\!1,a,n\!-\!1,n]\mathcal{A}_{n_R}^{(k_R)}\big(\widehat{a},a,\ldots,n\!-\!1,\widehat{n}\big),
$$

where the 'shifted' twistors $\widehat{a}$ and $\widehat{n}$ are defined according to:

$$
\begin{aligned}
\widehat{a} &\equiv (a\,a\!-\!1)\textstyle\bigcap(n\!-\!1\,n\,1) \equiv z_a\langle a\!-\!1\,n\!-\!1\,n\,1\rangle + z_{a-1}\langle n\!-\!1\,n\,1\,a\rangle;\\
\widehat{n} &\equiv (n\,n\!-\!1)\textstyle\bigcap(a\!-\!1\,a\,1) \equiv z_n\langle n\!-\!1\,a\!-\!1\,a\,1\rangle + z_{n-1}\langle a\!-\!1\,a\,1\,n\rangle.
\end{aligned} \tag{1.2}
$$

Because the recursion relations successively lower k (or n) and all MHV ($k\!=\!0$) amplitudes are 1, the only non-trivial functions that survive successive recursion via (1.1) are the so-called 'R-invariants' defined according to:

$$
R[a,b,c,d,e] \equiv \frac{\delta^{1\times4}\big(\eta_a\langle bcde\rangle+\eta_b\langle cdea\rangle+\eta_c\langle deab\rangle+\eta_d\langle eabc\rangle+\eta_e\langle abcd\rangle\big)}{\langle a\,b\,c\,d\rangle\langle b\,c\,d\,e\rangle\langle c\,d\,e\,a\rangle\langle d\,e\,a\,b\rangle\langle e\,a\,b\,c\rangle}, \tag{1.3}
$$

where $\{z_a,\ldots,z_e\}$ can be related to external momentum twistors through (possibly nested) shifts according to (1.2).

2 How to Do Supersymmetry (without Anticommutation)

Superfunctions multiply in the obvious way; and so a product of k R-invariants will take the form of $5k$ 4-brackets $\langle\cdots\rangle$ in the denominator with k (1×4) fermionic δ-functions in the numerator. Fermionic δ-functions also simply multiply, but because[1]

$$\int d\eta_a d\eta_b \; \delta(a_1\eta_a + b_1\eta_b)\delta(a_2\eta_a + b_2\eta_b) = (a_1 b_2 - a_2 b_1), \tag{2.1}$$

for anti-commuting variables, it is most natural to consider the $k\times n$ matrix 'C' of η-coefficients (with one row of C coming from each R-invariant) and think of these superfunctions to be in the form,

$$f(z)\, \delta^{k\times4}\big(C\cdot\eta\big) \equiv f(z)\prod_{I=1}^{4}\prod_{\alpha=1}^{k}\Big(\sum_{a=1}^{n} C_a^\alpha \eta_a^I\Big), \tag{2.2}$$

where $f(z)$ is an ordinary (bosonic) function, and C is a $k\times n$ matrix of ordinary functions. As illustrated in the simple case in (2.1), the reason for thinking of super-functions in this way is that all component functions—obtained by projecting out (via integration) $4k$ of the η's will result in simply multiplication of $f(z)$ by 4 ($k\times k$) determinants of the matrix C. To see this, notice that for each of the $SU(4)$-indices $I\in\{1,\ldots,4\}$, η-integration results in a determinant,

$$\int d\eta_{a_1}^I\cdots d\eta_{a_k}^I \;\; \prod_{\alpha=1}^{k}\Big(\sum_{a=1}^{n} C_a^\alpha \eta_a^I\Big) = (a_1\cdots a_k) \equiv \det\big(C_{a_1}^\alpha,\ldots,C_{a_k}^\alpha\big). \tag{2.3}$$

(This is a natural generalization of (2.1).) Thus, we see that the product of k R-invariants corresponds to a superfunction with $\binom{n}{k}^4$ bosonic components—corresponding to a choice of a ($k\times k$) minor of the matrix C for each $I\in\{1,\ldots,4\}$.

The point of this aside is that we will consider superfunctions (in MATHEMATICA) to correspond to a pair of purely bosonic data,

$$f(z)\, \delta^{k\times4}\big(C(z)\cdot\eta\big) \Longleftrightarrow \big\{f(z), C(z)\big\}, \tag{2.4}$$

from which all component functions can be extracted by multiplying $f(z)$ by 4 ($k\times k$) minors of the matrix $C(z)$. For superfunctions that are products of R-invariants, $f(z)$ is the product of their bosonic parts, and $C(z)$ is a matrix whose k rows are generated one-by-one from each R-invariant in the product.

---

[1] Recall that fermionic δ-functions are trivial: $\int d\eta\, \delta(f(\eta))\equiv\int d\eta\, f(\eta)$ for any Grassmann η.

3 Glossary of MATHEMATICA Functions to Know (and Love)

The following is a list of MATHEMATICA functions used in the live-demonstration of the implementation of tree-level BCFW (listed roughly in order of appearance).

- **Replace**[x,y] (same as 'x/.y') and **ReplaceRepeated**[x,y] (same as 'x//.y')
- **Rule**[x,y] (same as 'x->y') and **RuleDelayed**[x,y] (same as 'x:>y')
- **Set**[x,y] (same as 'x=y') and **SetDelayed**[x,y] (same as 'x:=y')
- **Which**[ ] (see also **If**[ ])
- **And**[x, y] (same as 'x&&y') and **Or**[x, y] (same as 'x||y')
- **Equal**[x, y] (same as 'x==y') and **SameQ**[x, y] (same as 'x===y'), etc.
- **With**[ ], (see also **Block**[ ], **Module**[ ])
- **Table**[function[*index*], {*index*,*indexRange*}] (see also **Sum**[ ], **Prod**[ ])
- **Join**[ ] (see also **Flatten**[ ])
- **Select**[*list*, testFunction[ ]]: selects those elements of *list* that are mapped to **True** under testFunction.
- **Apply**[function[ ], *exprn*] (same as 'function@@*exprn*'—see also 'f@@@*exprn*') e.g. f@@{x,y} returns f[x, y]; f@@@{{x,y},{z,w}} returns {f[x, y], f[z, w]}.
- **Map**[function[ ], *list*] (same as: 'function/@*list*')
- **Function**[{x}, function[x]] (same as 'function[#]&')
- **Total**[ ], **Times**[ ], **Plus**[ ], etc.
- **ReplacePart**[ ]
- **Range**[ ], **Partition**[ ], **RotateLeft**[ ], etc.
- **SparseArray**[ ] (and **Normal**[ ])
- **Thread**[ ] (especially in the context of **Thread**[**Rule**[ ]])
- **Cases**[ ]
- **Det**[ ] (please remember to never reinvent **Det**[ ]!)
- **RandomInteger**[ ]
- . . .

If any of the functions above are unfamiliar, consult MATHEMATICA's *Documentation Center* for more details (and illustrative examples). This can be accessed quickly by prepending a '?' to a function in the notebook interface (e.g. by typing '?**Partition**'). You will often find that many built-in functions have optional additional arguments (that can be extremely useful to know!).

Finally, if you think that '**Function**' is a bit odd or irrelevant, you are wrong: (especially when constructed using the '(#)&@' syntax) **Function** is one of the most useful programming constructs in MATHEMATICA—and the same goes for **Map** (/@) and **Apply** (@@ and @@@). Get to know and love their syntax and usage—it will dramatically improve your programming prowess.