Università Degli Studi
di Napoli Federico II

INFN

On the Numerical Evaluation of Loop Integrals for Higher-Order Calculations with `LINE`

9th International Workshop on High Precision for Hard Processes at the LHC

Turin, Campus Luigi Einaugi

Renato Maria Prisco

University of Naples Federico II & INFN - Naples

in collaboration with:

Jonathan Ronca
Francesco Tramontano

What is `LINE`?

`LINE` (**L**oop **I**ntegral **N**umerical **E**valuator) is a tool to compute **Loop Integrals** via **Differential Equations** (DEs)

Significant advancements in recent years:

- **`DiffExp`** → DEs via series expansion
- **`SeaSyde`** → DEs + complex masses
- **`AMFlow`** → DEs w.r.t. auxiliary mass (BCs at infinity)

great and robust **Mathematica packages**

multi-purpose high-level software → not tailored for this use case

license issue → not ideal for massive cluster computations

`LINE` aims to improve on these aspects

faster low-level language (**C**) → **optimal performance** (only-what-we-need approach)

open source (available in a few weeks) → suitable for massive cluster computations

- Method
- Implementation
- Examples

- Method
- Implementation
- Examples

Loop Integrals via Differential Equations

DEs can be used to propagate Loop Integrals across the phase-space

- use a starting point where **boundary conditions** can be obtained
- find DEs along the line connecting the initial and the **target point**
- solve DEs for different **numerical values of epsilon**
- **interpolate** epsilon orders

$$\vec{I}(s_i, t_i, \dots) \xrightarrow{\textit{propagation}} \vec{I}(s_f, t_f, \dots)$$

starting point **target point**

Loop Integrals via Differential Equations

DEs can be used to propagate Loop Integrals across the phase-space

- use a starting point where **boundary conditions** can be obtained
- find DEs along the line connecting the initial and the **target point**
- solve DEs for different **numerical values of epsilon**
- **interpolate** epsilon orders

$$\vec{I}(s_i, t_i, \dots) \xrightarrow{\textit{propagation}} \vec{I}(s_f, t_f, \dots)$$

starting point $\eta = 0$

target point $\eta = 1$

$$\begin{cases} s(\eta) = s_i + \eta(s_f - s_i) \\ t(\eta) = t_i + \eta(t_f - t_i) \\ \quad\vdots \end{cases}$$

$\eta \in [0, 1]$

line parameter

$$\partial_\eta = (s_f - s_i)\partial_s + (t_f - t_i)\partial_t + \dots$$

$$A(\eta) = (s_f - s_i)A_s + (t_f - t_i)A_t + \dots$$

Loop Integrals via Differential Equations

DEs can be used to propagate Loop Integrals across the phase-space

- use a starting point where **boundary conditions** can be obtained
- find DEs along the line connecting the initial and the **target point**
- solve DEs for different **numerical values of epsilon**
- **interpolate** epsilon orders

propagation

$$\vec{I}(s_i, t_i, \dots) \longrightarrow \vec{I}(s_f, t_f, \dots)$$

starting point $\eta = 0$

target point $\eta = 1$

$$\begin{cases} s(\eta) = s_i + \eta(s_f - s_i) \\ t(\eta) = t_i + \eta(t_f - t_i) \\ \quad\vdots \end{cases}$$

$\eta \in [0, 1]$

line parameter

$$\partial_\eta = (s_f - s_i)\partial_s + (t_f - t_i)\partial_t + \dots$$

$$A(\eta) = (s_f - s_i)A_s + (t_f - t_i)A_t + \dots$$

Loop Integrals via Differential Equations

DEs can be used to propagate Loop Integrals across the phase-space

- use a starting point where **boundary conditions** can be obtained
- find DEs along the line connecting the initial and the **target point**
- solve DEs for different **numerical values of epsilon**
- **interpolate** epsilon orders

AMFlow method
introduce **auxiliary mass** to
get BCs at infinity,
then propagate to zero mass

implemented in **`LINE`** with interface to **`Kira`**

$$\vec{I}(s_i, t_i, \dots) \xrightarrow{\textit{propagation}} \vec{I}(s_f, t_f, \dots)$$

starting point $\eta = 0$

target point $\eta = 1$

$$\begin{cases} s(\eta) = s_i + \eta(s_f - s_i) \\ t(\eta) = t_i + \eta(t_f - t_i) \\ \vdots \end{cases}$$

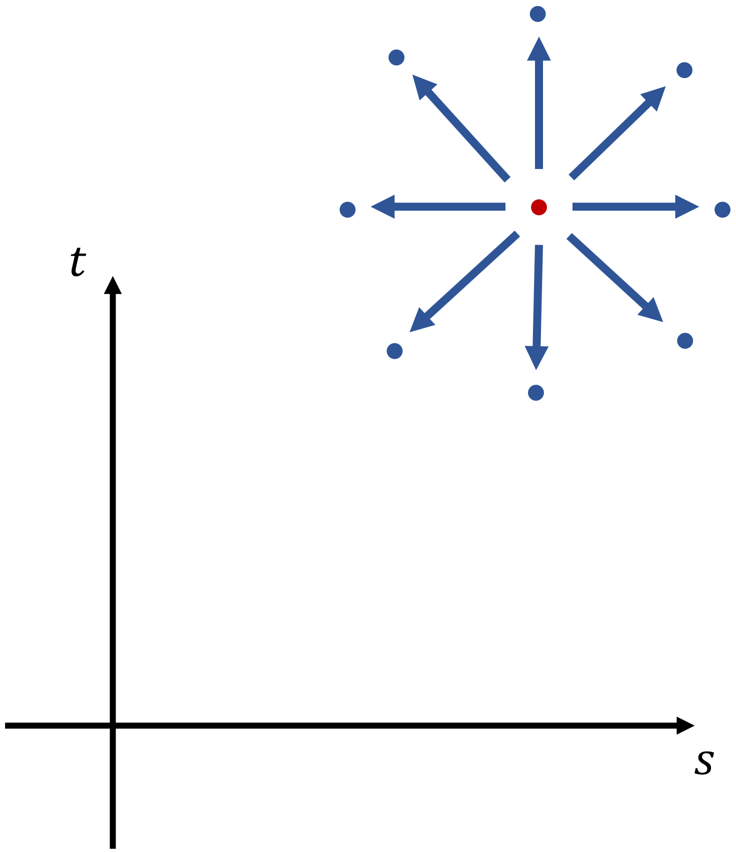
$\eta \in [0, 1]$
line parameter

$$\partial_\eta = (s_f - s_i)\partial_s + (t_f - t_i)\partial_t + \dots$$

$$A(\eta) = (s_f - s_i)A_s + (t_f - t_i)A_t + \dots$$

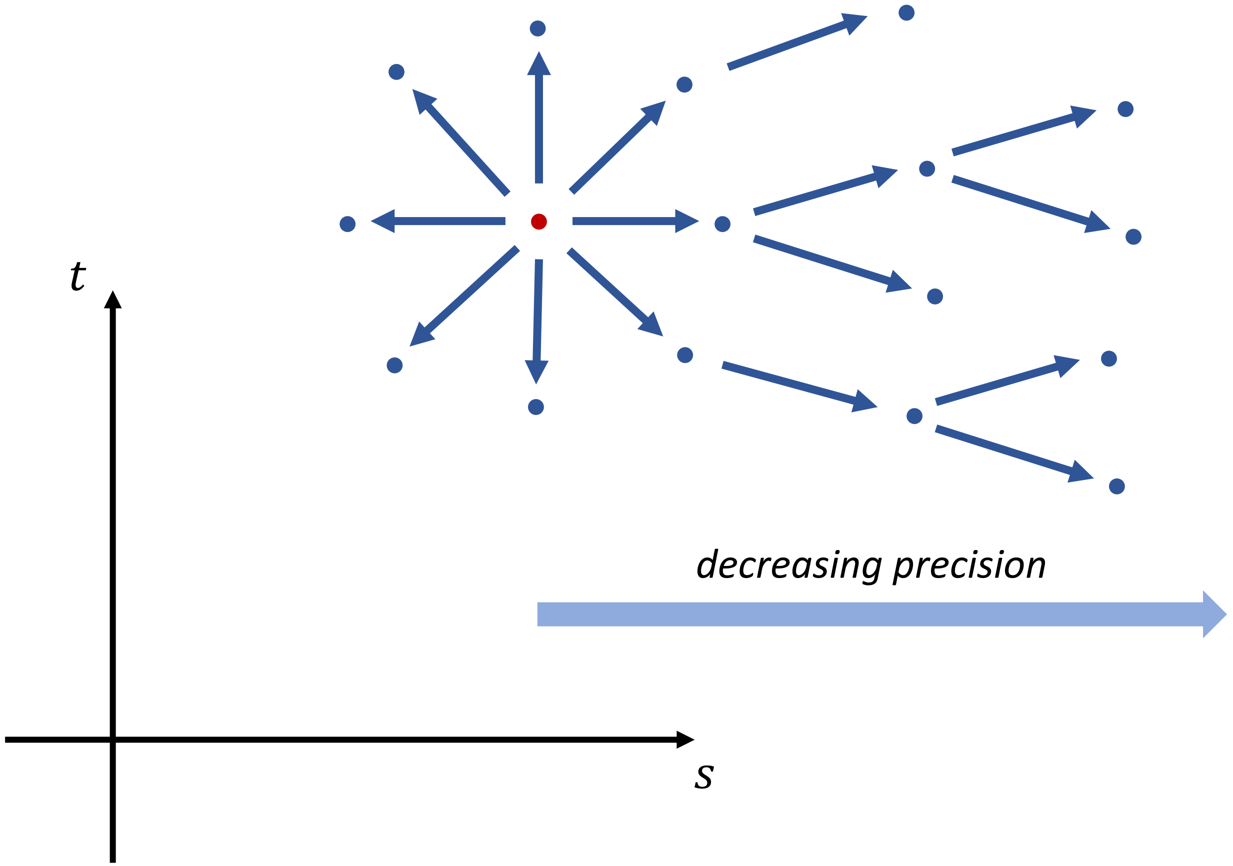
Loop Integrals via Differential Equations

Get BCs at one point, then propagate to many other points



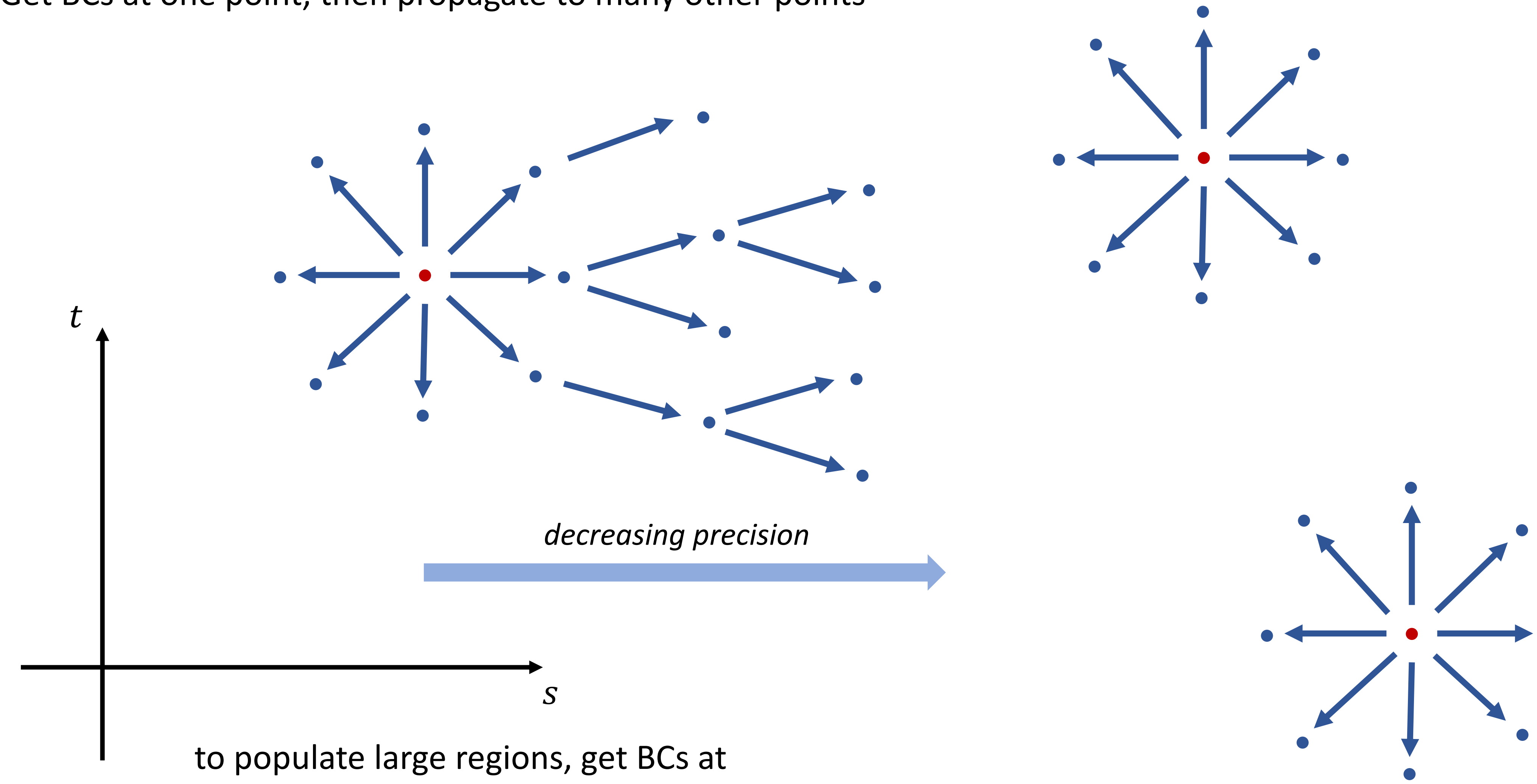
Loop Integrals via Differential Equations

Get BCs at one point, then propagate to many other points



Loop Integrals via Differential Equations

Get BCs at one point, then propagate to many other points

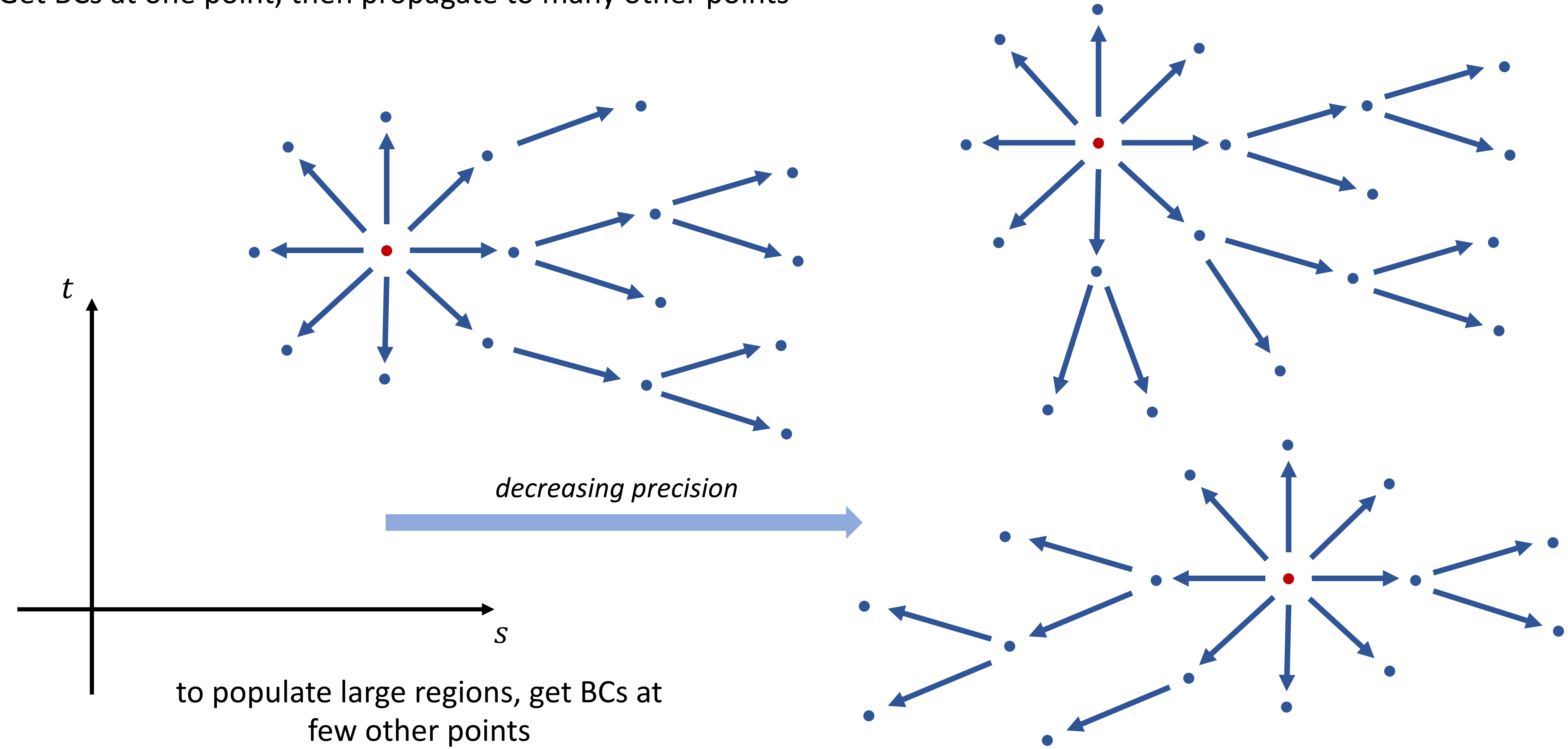


decreasing precision

to populate large regions, get BCs at
few other points

Loop Integrals via Differential Equations

Get BCs at one point, then propagate to many other points



to populate large regions, get BCs at few other points

Loop Integrals via Differential Equations

Introducing additional parameters can be computationally **expensive**, especially for complicated topologies that already have several parameters

AMFlow method

introduce **auxiliary mass** to get BCs at infinity, then propagate to zero mass

For integrals with **massive lines**, get **automated BCs** around **vanishing kinematics** using **Expansion by Region**

successful on few examples

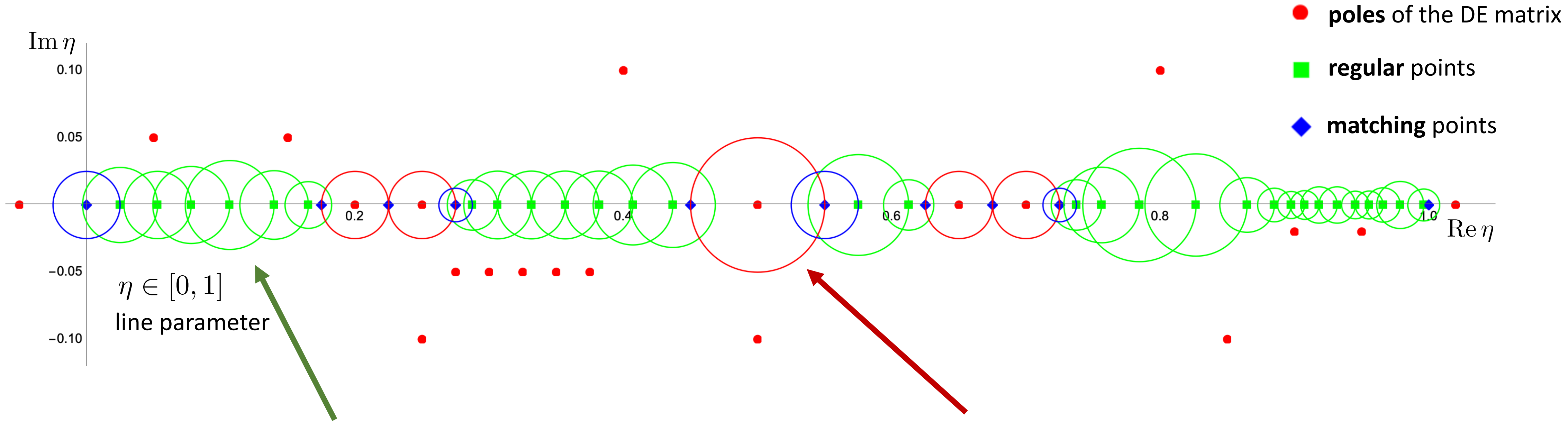
- vanishing kinematics → masses are seen as infinite
- Expansion by Regions predicts behaviour around such point
- force the solution to have such behaviour

work in progress

(under investigation)

Poles and Series Expansion

DEs have **poles** → series expansion within **radius of convergence**



Near regular points the solution has a Taylor expansion:

$$I(\eta) = \sum_{k=0}^{\infty} c_k \eta^k$$

ansatz around **regular** points

Cross singular points using:

max log power

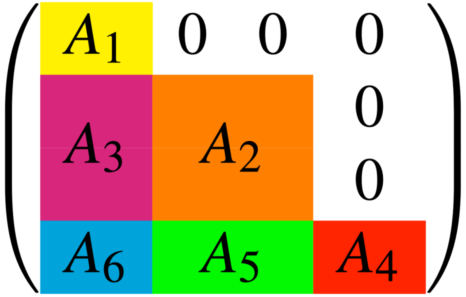
$$I(\eta) = \sum_{\lambda \in S} \eta^\lambda \sum_{l=0}^{L_\lambda} \sum_{k=0}^{\infty} c_{\lambda,l,k} \log^l(\eta) \eta^k$$

set of eigenvalues

ansatz around **regular-singular** points

Block Strategy

Exploit the **block lower triangular** structure of the DE matrix:



$$A = \begin{pmatrix} A_1 & 0 \quad 0 & 0 \\ A_3 & A_2 & \begin{matrix} 0 \\ 0 \end{matrix} \\ A_6 & A_5 & A_4 \end{pmatrix}$$

- solve one block at a time (much smaller problem)
- trade homogeneous DE for **non-homogeneous** ones

After solving $b-1$ blocks:

$$\partial_\eta \vec{I}_b = A_{b,b}(\eta)\vec{I}_b + \vec{Y}_b$$

$$\vec{Y}_b = (A_{b,1}, \ldots, A_{b,b-1}) \underbrace{\begin{pmatrix} \vec{I}_1 \\ \ldots \\ \vec{I}_{b-1} \end{pmatrix}}_{\text{known from previous blocks}}$$

solve **non-homogeneous** DEs around **regular-singular** points by **series expansion**

- Method
- **Implementation**
- Examples

Fractions of Polynomials

Need to manipulate symbolic expressions containing large **fractions of polynomials**:

- perform basic operations (sum, product...) → Least Common Multiple (**LCM**), simplifications
- evaluate **singular behaviour** around poles
- perform **shifts** $\eta \to \eta + \eta_0$

$\mathbb{N}\ \mathbb{Q}$ $\mathbb{R}$ $\mathbb{C}$

`gmp, mpfr, mpc`

for **abitrary precision**

functional, fast,
open source
and well maintained

C implementation tailored to our specific use case:

- store numerator as **coefficients**
- store denominator as **list of roots**
  (fast computation of LCM, simplifications, etc.)

coefficient

$$a_0 + a_1\eta + a_2\eta^2 + \dots$$

$$\eta^{m_0}(\eta - \eta_1)^{m_1}(\eta - \eta_2)^{m_2} \dots$$

root

multiplicity

Fractions of Polynomials

Need to manipulate symbolic expressions containing large **fractions of polynomials**:

- perform basic operations (sum, product...) → Least Common Multiple (**LCM**), simplifications
- evaluate **singular behaviour** around poles
- perform **shifts** $\eta \to \eta + \eta_0$

can we do better?

$\mathbb{N}\ \mathbb{Q}$ $\mathbb{R}$ $\mathbb{C}$

`gmp`, `mpfr`, `mpc`

for **abitrary precision**

functional, fast,
open source
and well maintained

C implementation tailored to our specific use case:

- store numerator as **coefficients**
- store denominator as **list of roots**
  (fast computation of LCM, simplifications, etc.)

coefficient

$$a_0 + a_1\eta + a_2\eta^2 + \ldots$$

$$\eta^{m_0}(\eta - \eta_1)^{m_1}(\eta - \eta_2)^{m_2}\ldots$$

root

multiplicity

Fractions of Polynomials

Need to manipulate symbolic expressions containing large **fractions of polynomials**:

- perform basic operations (sum, product...) → Least Common Multiple (**LCM**), simplifications
- evaluate **singular behaviour** around poles
- perform **shifts** $\eta \to \eta + \eta_0$

can we do better?

$\mathbb{N}\,\mathbb{Q}$ $\mathbb{R}$ $\mathbb{C}$

`gmp`, `mpfr`, `mpc`

for **abitrary precision**

functional, fast,
open source
and well maintained

C implementation tailored to our specific use case:

- store numerator as **coefficients**

coefficient

$$a_0 + a_1\eta + a_2\eta^2 + \ldots$$

- store denominator as **list of roots**
  (fast computation of LCM, simplifications, etc.)

$$\eta^{m_0}(\eta - \eta_1)^{m_1}(\eta - \eta_2)^{m_2}\ldots$$

root

multiplicity

roots typically appear more than once the matrix...

Fractions of Polynomials

$$\begin{pmatrix} \frac{p_{11}(\eta)}{\eta(\eta-42)^3} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{p_{n1}(\eta)}{\eta^2(\eta-42)^2(\eta-17)^4} & \cdots & \frac{p_{nn}(\eta)}{\eta(\eta-42)^3(\eta-10)^2} \end{pmatrix}$$

$$\Downarrow$$

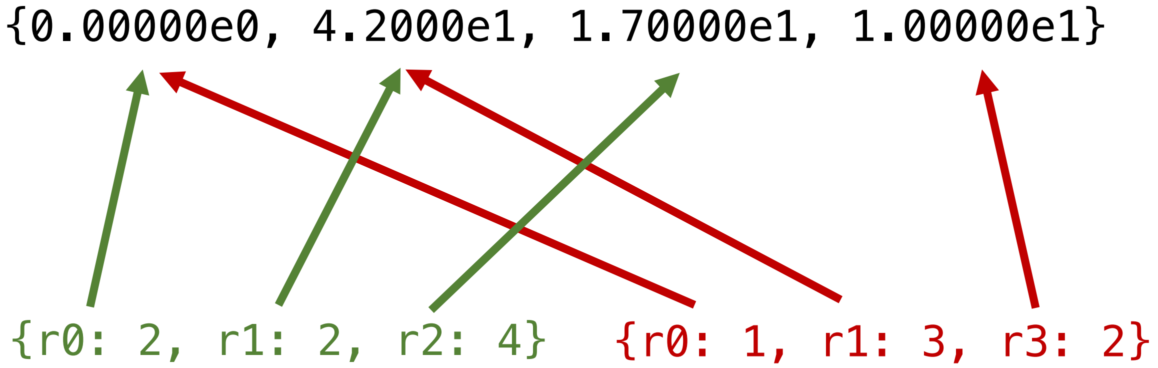
$$\begin{pmatrix} \texttt{\{r0: 1, r1: 3\}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \texttt{\{r0: 2, r1: 2, r2: 4\}} & \cdots & \texttt{\{r0: 1, r1: 3, r3: 2\}} \end{pmatrix}$$

LINE:

- DE matrices are scanned to find roots of the denominators (**numerically** with **arbitrary precision**)
- a list of **unique roots** is stored and updated
- a unique **label** is assigned to each root
- for each matrix element **only the labels are stored**

list of unique roots

(stored only once and accessed **only when necessary**)

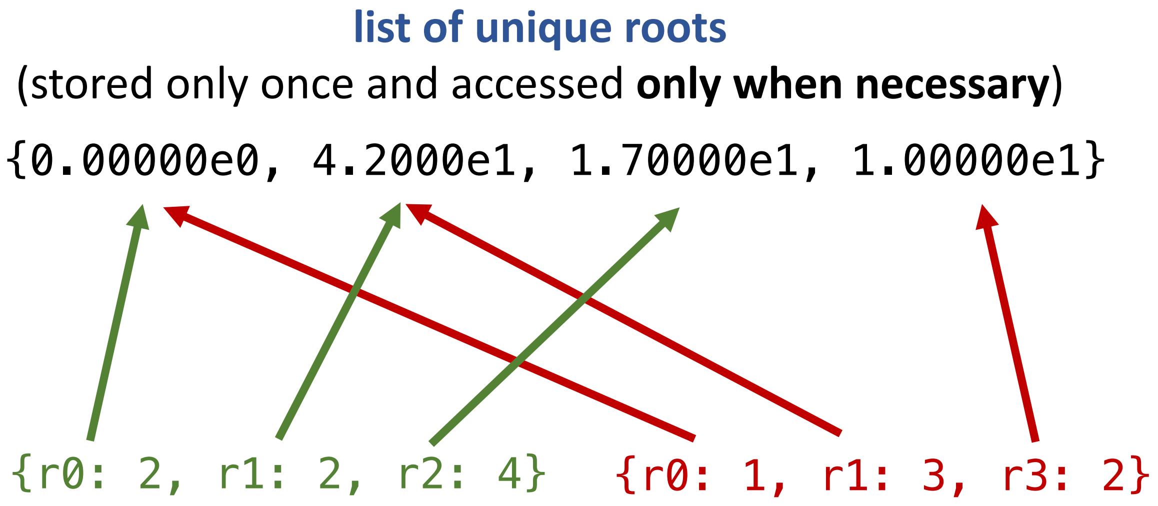


`{0.00000e0, 4.2000e1, 1.70000e1, 1.00000e1}`

`{r0: 2, r1: 2, r2: 4}` `{r0: 1, r1: 3, r3: 2}`

Fractions of Polynomials

$$\begin{pmatrix} \texttt{\{r0: 1, r1: 3\}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \texttt{\{r0: 2, r1:2, r2: 4\}} & \cdots & \texttt{\{r0: 1, r1: 3, r3: 2\}} \end{pmatrix}$$



list of unique roots

(stored only once and accessed **only when necessary**)

`{0.00000e0, 4.2000e1, 1.70000e1, 1.00000e1}`

`{r0: 2, r1: 2, r2: 4}` `{r0: 1, r1: 3, r3: 2}`

- sum: LCM only deals with integer labels

$$\frac{p_1(\eta)}{\eta^2(\eta-42)^2(\eta-17)^4} + \frac{p_2(\eta)}{\eta(\eta-42)^3(\eta-10)^2} = \frac{(\eta-42)(\eta-10)^2 p_1(\eta)+\eta\,p_2(\eta)(\eta-17)^4}{\eta^2(\eta-42)^3(\eta-17)^4(\eta-10)^2}$$

`LCM({r0: 2, r1: 2, r2: 4}, {r0: 1, r1: 3, r3: 2}) = {r0: 2, r1: 3, r2: 4, r3: 2}`

- shift $\eta \to \eta + \eta_0$: shit only numerators and list of roots $\{0,\ 42,\ 17,\ 10\} \ \to \{-\eta_0,\ 42-\eta_0,\ 17-\eta_0,\ 10-\eta_0\}$

Mathematical Expressions in C

How do we go from

```
2*(-4+d)^2*(10-7*d+d^2)*s^9*t^5*(3*s+4*t)*(3*(-3+d)*s+(-16+5*d)*t)-m2*s^7*t^4*(8-6*d+d^2)* (2*(-4150+3205*d-805*d^2+66*d^3)*s^4+(-42520+32724*d-
8219*d^2+675*d^3)*s^3*t+2*(-35500+27140*d-6743*d^2+532*d^3+3*d^4)*s^2*t^2+4*(-9914+7471*d-1782*d^2+117*d^3+4*d^4)*s*t^3+8*(-50-149*d+148*d^2-
43*d^3+4*d^4)*t^4)+262144*(-56+58*d-19*d^2+2*d^3)*m2^10*(s+t)*(4*(20-9*d+d^2)*s^5+(535-272*d+33*d^2)*s^4*t+(1265-753*d+135*d^2-7*d^3)*s^3*t^2+(1797-1281*d+309*d^2-
25*d^3)*s^2*t^3-2*(-638+525*d-144*d^2+13*d^3)*s*t^4+4*(71-70*d+21*d^2-2*d^3)*t^5)+4*(-2+d)*m2^2*s^6*t^3*((7640-6653*d+2015*d^2-243*d^3+9*d^4)*s^5+4*(33430-
33161*d+12130*d^2-1942*d^3+115*d^4)*s^4*t+(444340-442838*d+161309*d^2-24828*d^3+1165*d^4+36*d^5)*s^3*t^2+2*(249308-242356*d+83109*d^2-
10606*d^3+2*d^4+67*d^5)*s^2*t^3+2*(69640-49054*d+2605*d^2+5514*d^3-1535*d^4+122*d^5)*s*t^4+4*(-1048+7970*d-7887*d^2+3094*d^3-545*d^4+36*d^5)*t^5)+16*(-
2+d)*m2^3*s^5*t^2*(6*(3100-3465*d+1444*d^2-265*d^3+18*d^4)*s^6+(68620-79124*d+33975*d^2-6404*d^3+445*d^4)*s^5*t+(-29500+1205*d+15979*d^2-7473*d^3+1253*d^4-
72*d^5)*s^4*t^2-3*(69988-44332*d-1513*d^2+6687*d^3-1654*d^4+124*d^5)*s^3*t^3+(77798-240621*d+204195*d^2-74961*d^3+12691*d^4-814*d^5)*s^2*t^4-8*(-35409+56641*d-
35712*d^2+11089*d^3-1695*d^4+102*d^5)*s*t^5-4*(-8742+20263*d-15499*d^2+5397*d^3-887*d^4+56*d^5)*t^6)-65536*m2^9*(2*(-7760+11692*d-6838*d^2+1952*d^3-
273*d^4+15*d^5)*s^7+(-133880+204796*d-121754*d^2+35335*d^3-5021*d^4+280*d^5)*s^6*t-2*(230470-363039*d+225204*d^2-69833*d^3+11179*d^4-832*d^5+19*d^6)*s^5*t^2+(-
881296+1460192*d-974010*d^2+335937*d^3-63289*d^4+6178*d^5-244*d^6)*s^4*t^3+(-1117108+1966668*d-1416163*d^2+536687*d^3-113295*d^4+12661*d^5-586*d^6)*s^3*t^4-
2*(425080-786430*d+597375*d^2-239205*d^3+53345*d^4-6287*d^5+306*d^6)*s^2*t^5-4*(75164-145432*d+115125*d^2-47790*d^3+10987*d^4-1328*d^5+66*d^6)*s*t^6-8*(3976-
8038*d+6585*d^2-2802*d^3+655*d^4-80*d^5+4*d^6)*t^7)-64*m2^4*s^4*t*((-27440+43838*d-27405*d^2+8407*d^3-1267*d^4+75*d^5)*s^7+2*(-137060+215942*d-
132876*d^2+40069*d^3-5932*d^4+345*d^5)*s^6*t+(-830000+1325770*d-834559*d^2+262061*d^3-42123*d^4+3063*d^5-60*d^6)*s^5*t^2+(-1369568+2316650*d-
1582833*d^2+560776*d^3-108795*d^4+10962*d^5-448*d^6)*s^4*t^3-2*(1107084-1971420*d+1439151*d^2-554278*d^3+119240*d^4-13619*d^5+646*d^6)*s^3*t^4+(-
2689240+4874294*d-3633815*d^2+1433455*d^3-316573*d^4+37173*d^5-1814*d^6)*s^2*t^5-2*(613448-1148204*d+884122*d^2-359659*d^3+81686*d^4-9831*d^5+490*d^6)*s*t^6-
4*(25760-49326*d+38467*d^2-15660*d^3+3517*d^4-414*d^5+20*d^6)*t^7)+16384*m2^8*(4*(-3340+5218*d-3180*d^2+949*d^3-139*d^4+8*d^5)*s^8+(-168680+260686*d-
156935*d^2+46219*d^3-6677*d^4+379*d^5)*s^7*t+(-776760+1215342*d-744523*d^2+225335*d^3-34281*d^4+2243*d^5-28*d^6)*s^6*t^2+(-1750824+2832518*d-
1824311*d^2+597425*d^3-104127*d^4+9011*d^5-292*d^6)*s^5*t^3+(-2527592+4334208*d-3021960*d^2+1102421*d^3-222816*d^4+23735*d^5-1044*d^6)*s^4*t^4-2*(1257572-
2284780*d+1704803*d^2-671531*d^3+147648*d^4-17202*d^5+830*d^6)*s^3*t^5+(-1357816+2585562*d-2022013*d^2+832937*d^3-190839*d^4+23071*d^5-1150*d^6)*s^2*t^6-
2*(142736-285392*d+232778*d^2-99235*d^3+23358*d^4-2883*d^5+146*d^6)*s*t^7-4*(3976-8038*d+6585*d^2-2802*d^3+655*d^4-80*d^5+4*d^6)*t^8)+256*m2^5*s^3*(2*(-
3400+5400*d-3354*d^2+1022*d^3-153*d^4+9*d^5)*s^8+(-119920+186864*d-113586*d^2+33817*d^3-4943*d^4+284*d^5)*s^7*t+(-644040+1006058*d-614765*d^2+185243*d^3-
27921*d^4+1779*d^5-18*d^6)*s^6*t^2+(-1660112+2669004*d-1703084*d^2+549991*d^3-93801*d^4+7826*d^5-236*d^6)*s^5*t^3-2*(1458844-2455646*d+1670234*d^2-
590065*d^3+114572*d^4-11626*d^5+483*d^6)*s^4*t^4+(-3834288+6658564*d-4715934*d^2+1753646*d^3-362739*d^4+39719*d^5-1804*d^6)*s^3*t^5-2*(1286796-
2271580*d+1638513*d^2-621235*d^3+131070*d^4-14633*d^5+677*d^6)*s^2*t^6-4*(108724-179486*d+117200*d^2-38339*d^3+6476*d^4-507*d^5+12*d^6)*s*t^7+8*(-680-
6274*d+11075*d^2-7130*d^3+2215*d^4-336*d^5+20*d^6)*t^8)+4096*m2^7*s*(4*(-2180+3146*d-1742*d^2+465*d^3-60*d^4+3*d^5)*s^8+2*(-14200+20880*d-11748*d^2+3163*d^3-
407*d^4+20*d^5)*s^7*t+(203300-309420*d+181227*d^2-50405*d^3+6257*d^4-155*d^5-20*d^6)*s^6*t^2+(1010576-1573724*d+954516*d^2-282685*d^3+40814*d^4-2257*d^5-
8*d^6)*s^5*t^3+2*(986890-1627267*d+1075037*d^2-364353*d^3+66629*d^4-6194*d^5+226*d^6)*s^4*t^4+(2712280-4827004*d+3520554*d^2-1353979*d^3+290645*d^4-
33096*d^5+1564*d^6)*s^3*t^5+(2270132-4284872*d+3330633*d^2-1368343*d^3+313753*d^4-38077*d^5+1910*d^6)*s^2*t^6+16*(48454-97161*d+79799*d^2-34386*d^3+8206*d^4-
1029*d^5+53*d^6)*s*t^7+4*(18044-38504*d+33167*d^2-14795*d^3+3617*d^4-461*d^5+24*d^6)*t^8)-1024*m2^6*s^2*(8*(-2230+3441*d-2069*d^2+609*d^3-88*d^4+5*d^5)*s^8+(-
166360+254642*d-151519*d^2+44024*d^3-6265*d^4+350*d^5)*s^7*t+(-596680+937046*d-578117*d^2+177600*d^3-27977*d^4+2018*d^5-42*d^6)*s^6*t^2+(-1084336+1798006*d-
1197961*d^2+411803*d^3-77217*d^4+7505*d^5-296*d^6)*s^5*t^3-2*(661356-1160694*d+833066*d^2-314989*d^3+66520*d^4-7468*d^5+349*d^6)*s^4*t^4+(-775816+1362616*d-
978084*d^2+369317*d^3-77690*d^4+8657*d^5-400*d^6)*s^3*t^5+2*(310184-624798*d+520701*d^2-229750*d^3+56511*d^4-7331*d^5+391*d^6)*s^2*t^6+2*(351912-
727320*d+616792*d^2-274703*d^3+67760*d^4-8775*d^5+466*d^6)*s*t^7+4*(23880-58276*d+55626*d^2-26933*d^3+7046*d^4-951*d^5+52*d^6)*t^8)
```

to actual polynomial coefficients?

Mathematical Expressions in C

How do we go from

```
2*(-4+d)^2*(10-7*d+d^2)*s^9*t^5*(3*s+4*t)*(3*(-3+d)*s+(-16+5*d)*t)[illegible]m2*s^7*t^4*(8[illegible]6*[illegible]+d^2)* (2*(-4150+[illegible]205*d-805*d^2+66*d^3)*s^4+(-42520+32724*d-
8219*d^2+675*d^3)*s^3*t+2*(-35500+27140*d-6743*d^2+532*d^3+3*d^4)*s^2*t^2+[illegible]914+7471*d-1782*d^2+117*d^3+4*d^4)*s*t^3+8*(-50-149*d+148*d^2-
43*d^3+4*d^4)*t^4)+262144*(-56+58*d-19*d^2+2*d^3)*m2^10*(s+t)*(4*(20-9*d+d^2)*s^5+(535-272*d+33*d^2)*s^4*t+(1265-753*d+135*d^2-7*d^3)*s^3*t^2+(1797-1281*d+309*d^2-
25*d^3)*s^2*t^3-2*(-638+525*d-144*d^2+13*d^3)*s*t^4+4*(71-70*d+21*d^2-2*d^3)*t^5)+4[illegible]m2[illegible]s^6*t^3*((7640-6653*d+2015*d^2-243*d^3+9*d^4)*s^5+4*(33430-
33161*d+12130*d^2-1942*d^3+115*d^4)*s^4*t+(444340-442838*d+161309*d^2-24828*d^3[illegible]65*d^4+3[illegible])*s^3*t^2+2*(249308-242356*d+83109*d^2-
10606*d^3+2*d^4+67*d^5)*s^2*t^3+2*(69640-49054*d+2605*d^2+5514*d^3-1535*d^4+122*d^5)*s*t^4+4*[illegible]+7970*d-7887*d^2+3094*d^3-545*d^4+36*d^5)*t^5)+16*(-
2+d)*m2^3*s^5*t^2*(6*(3100-3465*d+1444*d^2-265*d^3+18*d^4)*s^6+(68620-79124*d+33975*d^2-6404*d^3[illegible]d^4)*s^5*t+(-29500+1205*d+15979*d^2-7473*d^3+1253*d^4-
72*d^5)*s^4*t^2-3*(69988-44332*d-1513*d^2+6687*d^3-1654*d^4+124*d^5)*s^3*t^3+(77798-240621*d+204195[illegible]4961*d^3+12691*d^4-814*d^5)*s^2*t^4-8*(-35409+56641*d-
35712*d^2+11089*d^3-1695*d^4+102*d^5)*s*t^5-4*(-8742+20263*d-15499*d^2+5397*d^3-887*d^4+56*d^5)*t^6)-6[illegible]6*m2^9*(2*(-7760+11692*d-6838*d^2+1952*d^3-
273*d^4+15*d^5)*s^7+(-133880+204796*d-121754*d^2+35335*d^3-5021*d^4+280*d^5)*s^6*t-2*(230470-363039*d+225204*d^2-69833*d^3+11179*d^4-832*d^5+19*d^6)*s^5*t^2+(-
881296+1460192*d-974010*d^2+335937*d^3-63289*d^4+6178*d^5-244*d^6)*s^4*t^3+(-1117108+1966668*d-1416163*d^2+536687*d^3-113295*d^4+12661*d^5-586*d^6)*s^3*t^4-
2*(425080-786430*d+597375*d^2-239205*d^3+53345*d^4-6287*d^5+306*d^6)*s^2*t^5-4*(75164-145432*d+115125*d^2-47790*d^3+10987*d^4-1328*d^5+66*d^6)*s*t^6-8*(3976-
8038*d+6585*d^2-2802*d^3+655*d^4-80*d^5+4*d^6)*t^7)-64*m2^4*s^4*t*((-27440+43838*d-27405*d^2+8407*d^3-1267*d^4+75*d^5)*s^7+2*(-137060+215942*d-
132876*d^2+40069*d^3-5932*d^4+345*d^5)*s^6*t+(-830000+1325770*d-834559*d^2+262061*d^3-42123*d^4+3063*d^5-60*d^6)*s^5*t^2+(-1369568+2316650*d-
1582833*d^2+560776*d^3-108795*d^4+10962*d^5-448*d^6)*s^4*t^3-2*(1107084-1971420*d+1439151*d^2-554278*d^3+119240*d^4-13619*d^5+646*d^6)*s^3*t^4+(-
2689240+4874294*d-3633815*d^2+1433455*d^3-316573*d^4+37173*d^5-1814*d^6)*s^2*t^5-2*(613448-1148204*d+884122*d^2-359659*d^3+81686*d^4-9831*d^5+490*d^6)*s*t^6-
4*(25760-49326*d+38467*d^2-15660*d^3+3517*d^4-414*d^5+20*d^6)*t^7)+16384*m2^8*(4*(-3340+5218*d-3180*d^2+949*d^3-139*d^4+8*d^5)*s^8+(-168680+260686*d-
156935*d^2+46219*d^3-6677*d^4+379*d^5)*s^7*t+(-776760+1215342*d-744523*d^2+225335*d^3-34281*d^4+2243*d^5-28*d^6)*s^6*t^2+(-1750824+2832518*d-
1824311*d^2+597425*d^3-104127*d^4+9011*d^5-292*d^6)*s^5*t^3+(-2527592+4334208*d-3021960*d^2+1102421*d^3-222816*d^4+23735*d^5-1044*d^6)*s^4*t^4-2*(1257572-
2284780*d+1704803*d^2-671531*d^3+147648*d^4-17202*d^5+830*d^6)*s^3*t^5+(-1357816+2585562*d-2022013*d^2+832937*d^3-190839*d^4+23071*d^5-1150*d^6)*s^2*t^6-
2*(142736-285392*d+232778*d^2-99235*d^3+23358*d^4-2883*d^5+146*d^6)*s*t^7-4*(3976-8038*d+6585*d^2-2802*d^3+655*d^4-80*d^5+4*d^6)*t^8)+256*m2^5*s^3*(2*(-
3400+5400*d-3354*d^2+1022*d^3-153*d^4+9*d^5)*s^8+(-119920+186864*d-113586*d^2+33817*d^3-4943*d^4+284*d^5)*s^7*t+(-644040+1006058*d-614765*d^2+185243*d^3-
27921*d^4+1779*d^5-18*d^6)*s^6*t^2+(-1660112+2669004*d-1703084*d^2+549991*d^3-93801*d^4+7826*d^5-236*d^6)*s^5*t^3-2*(1458844-2455646*d+1670234*d^2-
590065*d^3+114572*d^4-11626*d^5+483*d^6)*s^4*t^4+(-3834288+6658564*d-4715934*d^2+1753646*d^3-362739*d^4+39719*d^5-1804*d^6)*s^3*t^5-2*(1286796-
2271580*d+1638513*d^2-621235*d^3+131070*d^4-14633*d^5+677*d^6)*s^2*t^6-4*(108724-179486*d+117200*d^2-38339*d^3+6476*d^4-507*d^5+12*d^6)*s*t^7+8*(-680-
6274*d+11075*d^2-7130*d^3+2215*d^4-336*d^5+20*d^6)*t^8)+4096*m2^7*s*(4*(-2180+3146*d-1742*d^2+465*d^3-60*d^4+3*d^5)*s^8+2*(-14200+20880*d-11748*d^2+3163*d^3-
407*d^4+20*d^5)*s^7*t+(203300-309420*d+181227*d^2-50405*d^3+6257*d^4-155*d^5-20*d^6)*s^6*t^2+(1010576-1573724*d+954516*d^2-282685*d^3+40814*d^4-2257*d^5-
8*d^6)*s^5*t^3+2*(986890-1627267*d+1075037*d^2-364353*d^3+66629*d^4-6194*d^5+226*d^6)*s^4*t^4+(2712280-4827004*d+3520554*d^2-1353979*d^3+290645*d^4-
33096*d^5+1564*d^6)*s^3*t^5+(2270132-4284872*d+3330633*d^2-1368343*d^3+313753*d^4-38077*d^5+1910*d^6)*s^2*t^6+16*(48454-97161*d+79799*d^2-34386*d^3+8206*d^4-
1029*d^5+53*d^6)*s*t^7+4*(18044-38504*d+33167*d^2-14795*d^3+3617*d^4-461*d^5+24*d^6)*t^8)-1024*m2^6*s^2*(8*(-2230+3441*d-2069*d^2+609*d^3-88*d^4+5*d^5)*s^8+(-
166360+254642*d-151519*d^2+44024*d^3-6265*d^4+350*d^5)*s^7*t+(-596680+937046*d-578117*d^2+177600*d^3-27977*d^4+2018*d^5-42*d^6)*s^6*t^2+(-1084336+1798006*d-
1197961*d^2+411803*d^3-77217*d^4+7505*d^5-296*d^6)*s^5*t^3-2*(661356-1160694*d+833066*d^2-314989*d^3+66520*d^4-7468*d^5+349*d^6)*s^4*t^4+(-775816+1362616*d-
978084*d^2+369317*d^3-77690*d^4+8657*d^5-400*d^6)*s^3*t^5+2*(310184-624798*d+520701*d^2-229750*d^3+56511*d^4-7331*d^5+391*d^6)*s^2*t^6+2*(351912-
727320*d+616792*d^2-274703*d^3+67760*d^4-8775*d^5+466*d^6)*s*t^7+4*(23880-58276*d+55626*d^2-26933*d^3+7046*d^4-951*d^5+52*d^6)*t^8)
```

```
m2*s^7*t^4*(8-6*d+d^2)
```

string representing a **mathematical expression**

to actual polynomial coefficients?

Mathematical Expressions in C

- **parse** and convert to a representation of a mathematical expression
- need to perform **basic operations** such as product expansions, substitutions, ...
- no need for complicated operations as in general purpose tools and packages

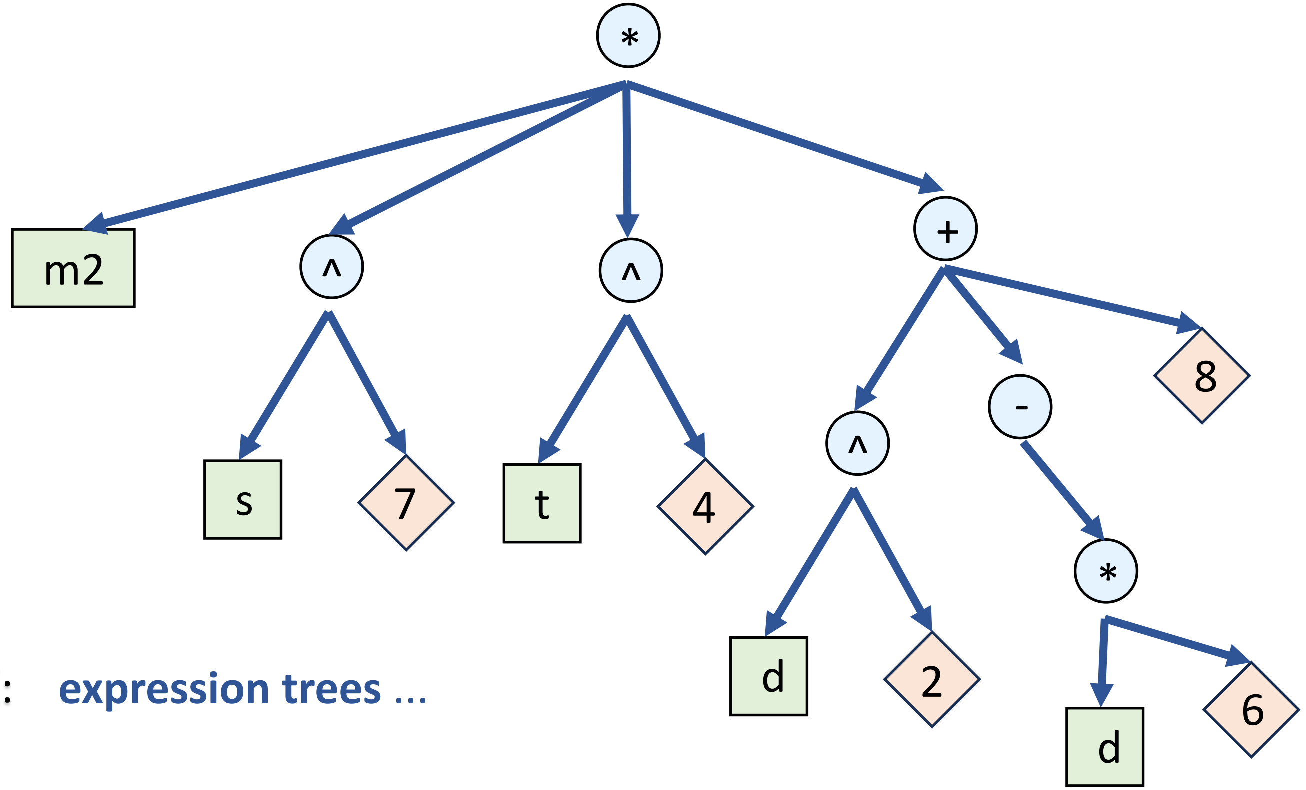
m2*s^7*t^4*(8-6*d+d^2)

string representing a **mathematical expression**

very powerful and useful tools, but can be unnecessarily slow

look for something lighter

build **from scratch** implementing **only what we need**



LINE: **expression trees ...**

Mathematical Expressions in C

- **parse** and convert to a representation of a mathematical expression
- need to perform **basic operations** such as product expansions, substitutions, ...
- no need for complicated operations as in general purpose tools and packages

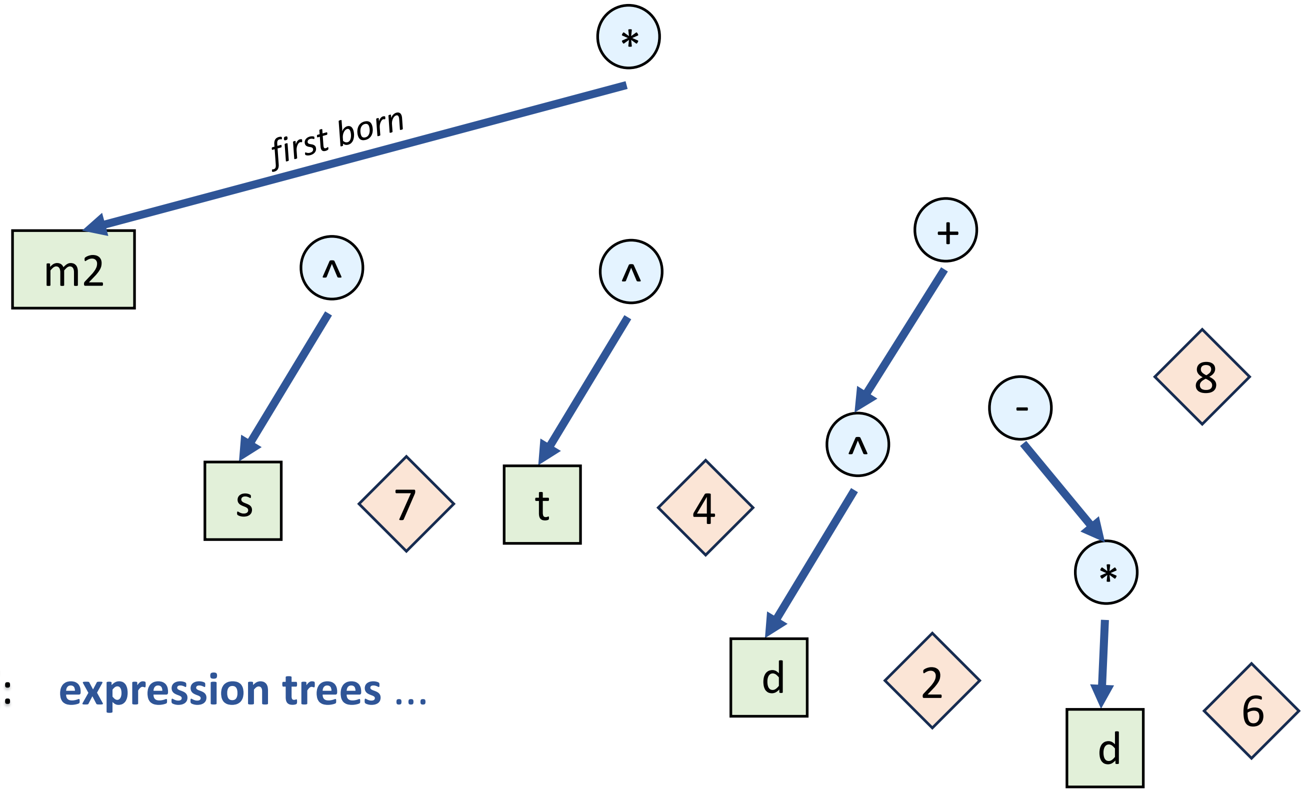
m2*s^7*t^4*(8-6*d+d^2)

string representing a
mathematical expression

very powerful and useful tools,
but can be unnecessarily slow

look for something lighter

build **from scratch** implementing
only what we need



LINE: **expression trees ...**

Mathematical Expressions in C

- **parse** and convert to a representation of a mathematical expression
- need to perform **basic operations** such as product expansions, substitutions, ...
- no need for complicated operations as in general purpose tools and packages

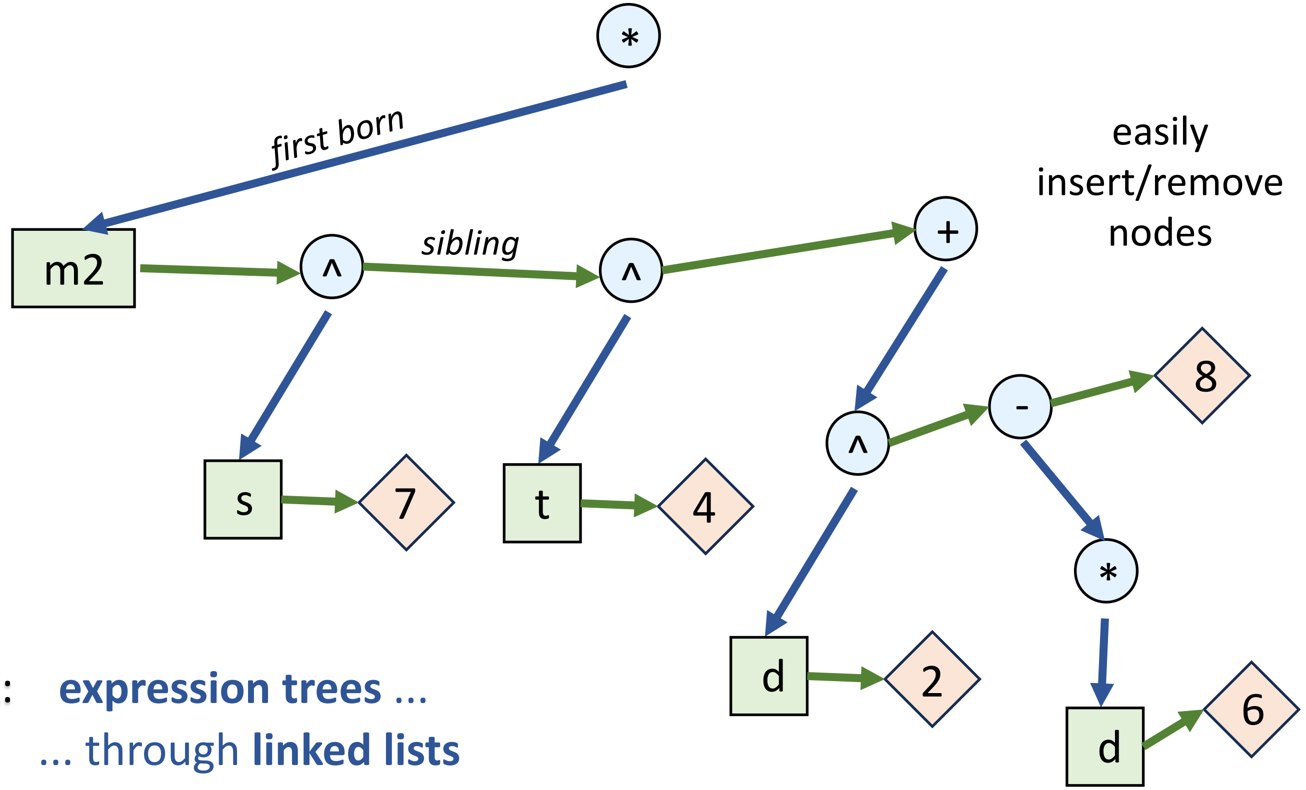
m2*s^7*t^4*(8-6*d+d^2)

string representing a **mathematical expression**

very powerful and useful tools, but can be unnecessarily slow

look for something lighter

build **from scratch** implementing **only what we need**



LINE: **expression trees** ...
... through **linked lists**

Analytic Continuation

Around **branch points** → infinite solutions matching BCs (with different branch cuts)

$$I(\eta) = \sum_{\lambda \in S} \eta^{\lambda} \sum_{l=0}^{L_\lambda} \sum_{k=0}^{\infty} c_{\lambda,l,k} \boxed{\log^l(\eta)} \eta^k$$

logarithms give a **branch cut** to the solution!

ansatz around **regular-singular** points

we must decide where to place the branch cut ➡ place it according to the **Feynman prescription**

$$\begin{cases} s_1(\eta) = s_{1,i} + \eta(s_{1,f} - s_{1,i}) \\ s_2(\eta) = s_{2,i} + \eta(s_{2,f} - s_{1,i}) \\ \quad \vdots \\ m_1^2(\eta) = m_{1,i}^2 + \eta(m_{1,f}^2 - m_{1,i}^2) \\ m_2^2(\eta) = m_{2,i}^2 + \eta(m_{2,f}^2 - m_{1,i}^2) \\ \quad \vdots \end{cases}$$

For any given **Cutkosky cut**, consider:

$$c_i, c'_j \in \{0, \pm 1\}$$

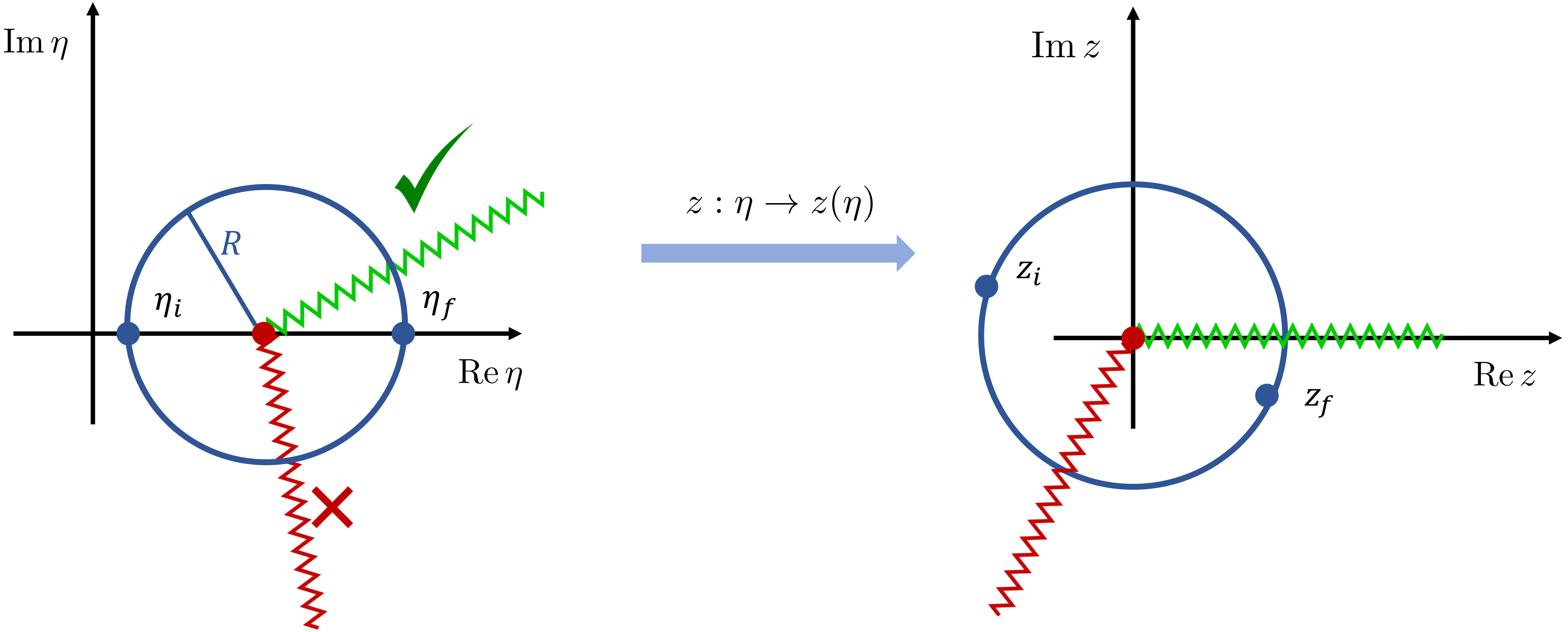
$$z = \underbrace{c_1 s_1 + c_2 s_2 + \cdots}_{\text{invariants flowing through the cut}} - \underbrace{(c'_1 m_1 + c'_2 m_2 + \cdots)^2}_{\text{masses of cut propagators}} = z(\eta)$$

curve in the z-complex plane

$z = 0$ **branch point** $z > 0$ **branch cut**

Analytic Continuation

The correct branch-cut in the η-plane is mapped to $z > 0$



If masses are fixed, the map is linear and there are no complications

Analytic Continuation

BUT when **squared masses** are **linearly varied** branch cuts in the η-plane can get **complicated shapes!**

$$z(\eta) = c_1 s_1(\eta) + c_2 s_2(\eta) + \cdots$$
$$- \left[c_1' \sqrt{m_{1,i}^2 + \eta(m_{1,f}^2 - m_{1,i}^2)} + c_2' \sqrt{m_{2,i}^2 + \eta(m_{2,f}^2 - m_{2,i}^2)} + \cdots \right]^2$$

$$\begin{cases} m_1^2(\eta) = m_{1,i}^2 + \eta(m_{1,f}^2 - m_{1,i}^2) \\ m_2^2(\eta) = m_{2,i}^2 + \eta(m_{2,f}^2 - m_{1,i}^2) \\ \qquad \vdots \end{cases}$$

Varying **linear masses** instead:

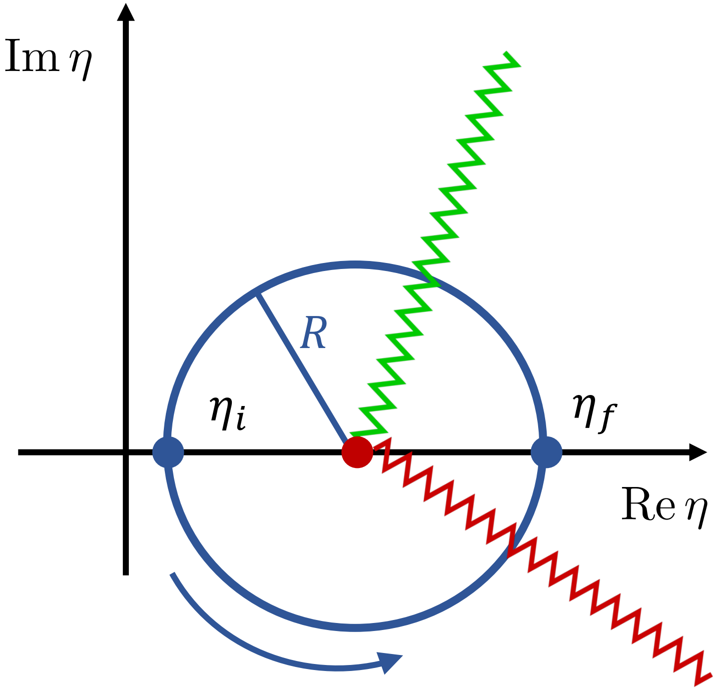
$$z(\eta) = c_1 s_1(\eta) + c_2 s_2(\eta) + \cdots$$
$$- [c_1' (m_{1,i} + \eta(m_{1,f} - m_{1,i})) + c_2' (m_{2,i} + \eta(m_{2,f} - m_{2,i})) + \cdots]^2$$

the map is **quadratic** → much **easier to handle**

$$\begin{cases} m_1(\eta) = m_{1,i} + \eta(m_{1,f} - m_{1,i}) \\ m_2(\eta) = m_{2,i} + \eta(m_{2,f} - m_{1,i}) \\ \qquad \vdots \end{cases}$$

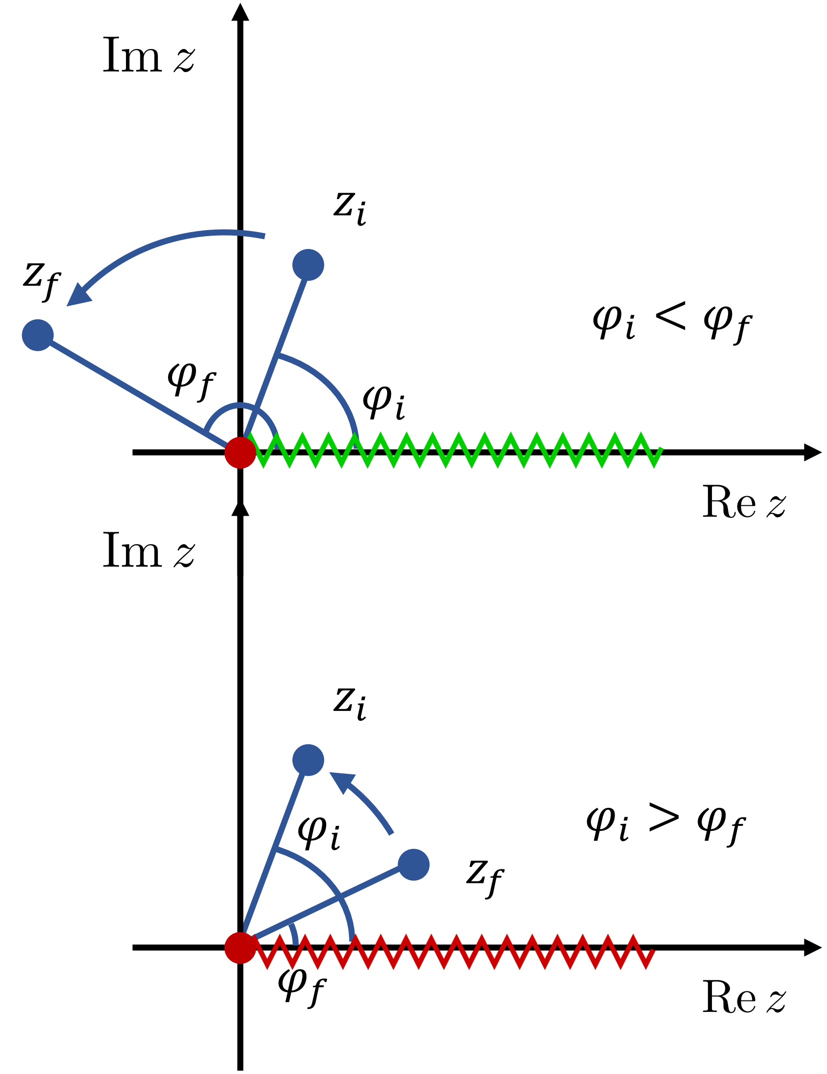
Analytic Continuation

Only need to know whether the branch cut is in the upper or lower half-plane



η and z rotate
in the same direction

$z : \eta \to z(\eta)$

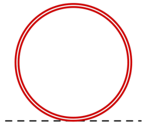


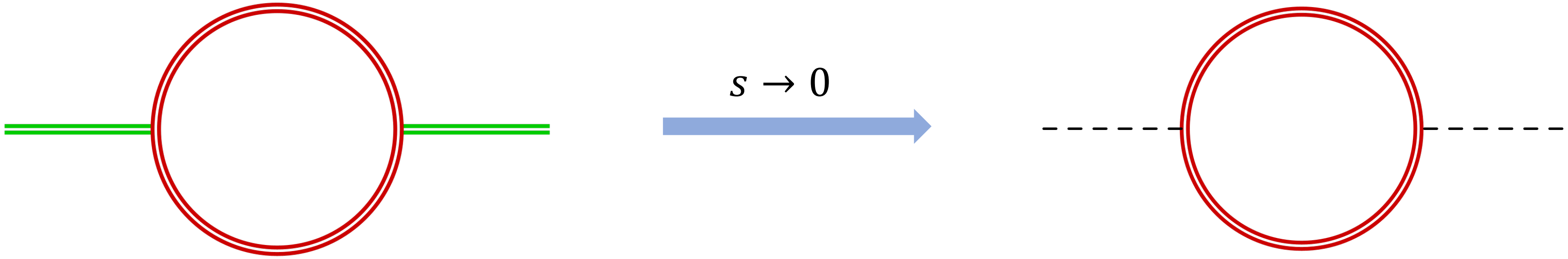
Is the branch cut, say, in the lower half-plane?

Imagine to rotate η counter-clockwise from η_i to η_f →
the branch cut is crossed $\Leftrightarrow$ z crosses the positive real axis $\Leftrightarrow$ $\varphi_i > \varphi_f$

Automated Boundary Conditions

1-loop massive bubble in the limit of **vanishing kinematics** → only **tadpole** is needed $A = -m^{2(1-\varepsilon)}\Gamma(\varepsilon - 1)$





$s \to 0$

know **regular** contribution

$$m^{-2\varepsilon}(\varepsilon-1)\Gamma(\varepsilon-1) = -\frac{(\varepsilon-1)}{m^2}A$$

can be obtained from the DE

DE for the bubble

regular

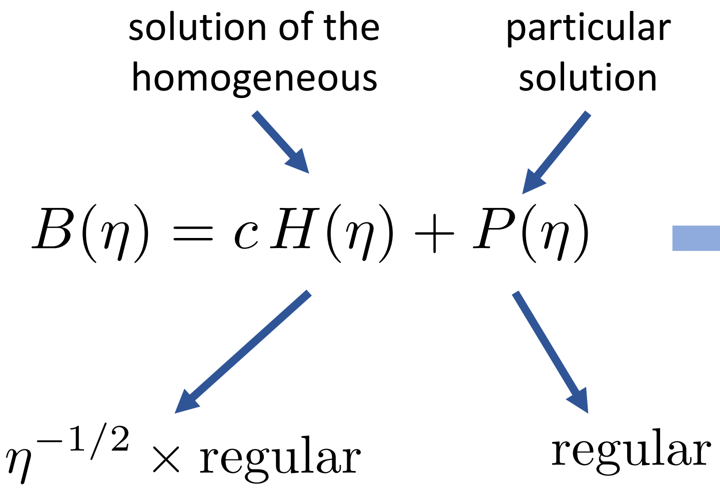
$$\partial_\eta B(\eta) = \frac{f_{B,B}(\eta)}{\eta}B(\eta) + \frac{f_{B,A}(\eta)}{\eta}A \qquad s \propto \eta$$

must be regular

$$0 = \frac{f_{B,B}(0)}{\eta}B(0) + \frac{f_{B,A}(0)}{\eta}A \quad \Longrightarrow \quad B(0) = -\frac{f_{B,A}(0)}{f_{B,B}(0)}A$$

Automated Boundary Conditions

The implementation is even simpler



solution of the homogeneous

particular solution

$$B(\eta) = c\, H(\eta) + P(\eta)$$

$\eta^{-1/2} \times \text{regular}$

regular

general solution

only $c = 0$ is allowed
to have a regular solution

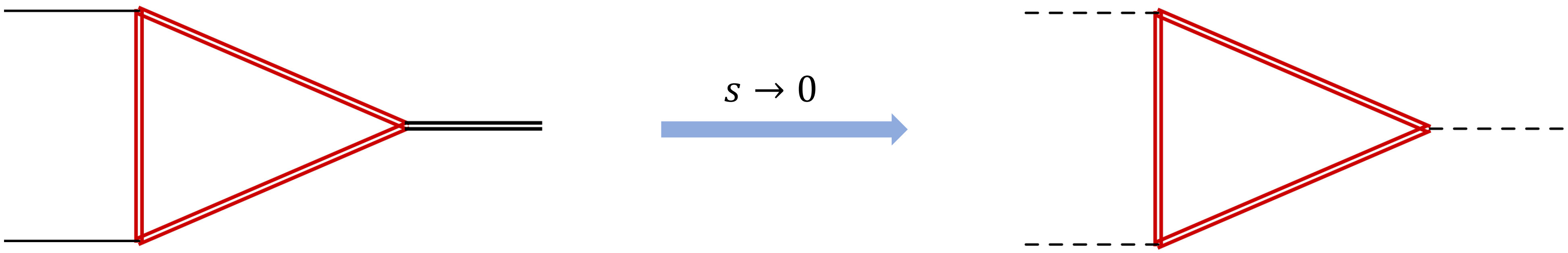
keep just the
particular solution

$$B(\eta) = P(\eta)$$

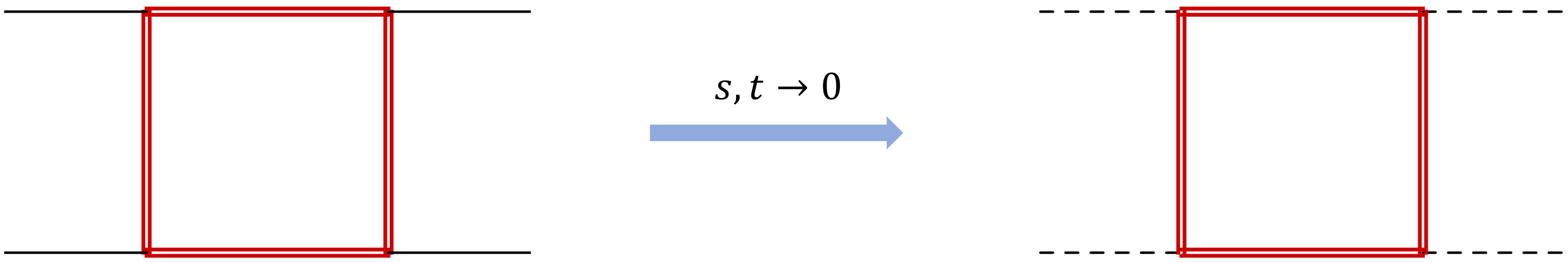
the DE automatically selects, as particular solution, the **unique regular solution**

Automated Boundary Conditions

Analogous for triangle and box



$$-m^{-2(1+\varepsilon)}\frac{\varepsilon}{2}(\varepsilon-1)\Gamma(\varepsilon-1)=\frac{\varepsilon(\varepsilon-1)}{2m^4}A$$

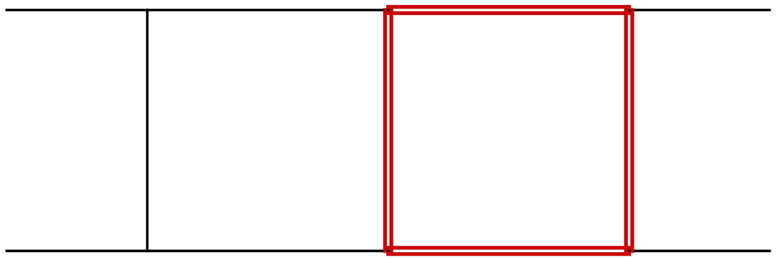


$$m^{-2(2+\varepsilon)}\frac{\varepsilon}{6}(\varepsilon^2-1)\Gamma(\varepsilon-1)=-\frac{\varepsilon(\varepsilon^2-1)}{6m^6}A$$

implementation: keep just the particular solution

Automated Boundary Conditions

A more involved example



loop momenta can be:

- **small** (S) → $k_i \ll m$
- **large** (L) → $k_i \sim m$

simple rules for propagators

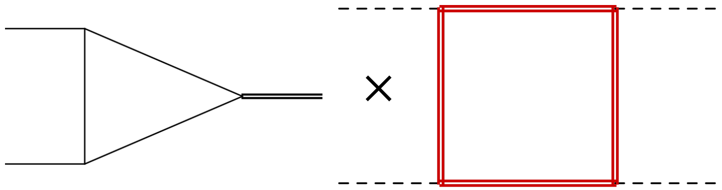
$$(k+p)^2 - m^2 \xrightarrow{S} -m^2 \qquad (k+p)^2 - m^2 \xrightarrow{L} k^2 - m^2$$

$$(k+p)^2 \xrightarrow{S} (k+p)^2 \qquad (k+p)^2 \xrightarrow{L} k^2$$

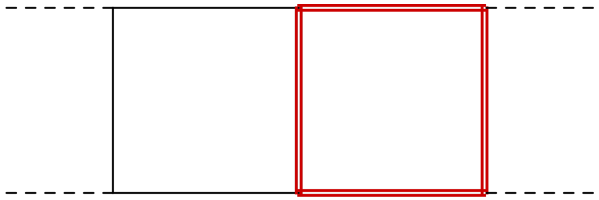
multiple regions:

k_1	k_2	$k_1 + k_2$	
S	S	S	scaleless = 0
S	L	L	→
L	S	L	scaleless = 0
L	L	S	scaleless = 0
L	L	L	→

$\eta^{-\varepsilon-1} \times \text{regular}$



regular

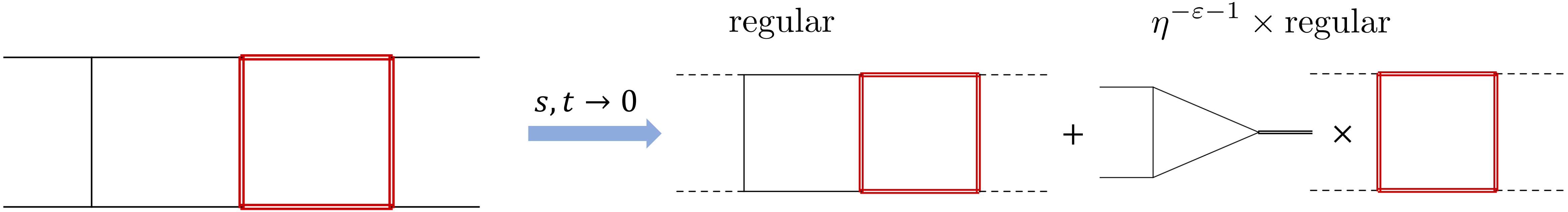


not need to know their expression!

only need the **exponent** of the regions

Automated Boundary Conditions

In each block of the DEs



Solve in a **Fuchsian basis**, then transform back

$$\vec{I}(\eta) = c_1 \eta^{\lambda_1 - n_1} \vec{h}_1(\eta) + c_2 \eta^{\lambda_2 - n_2} \vec{h}_2(\eta) + \cdots + \vec{p}(\eta)$$

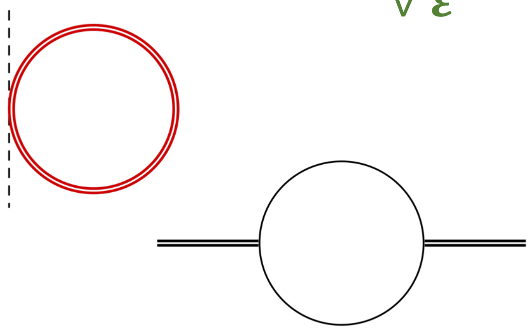
$$s, t \propto \eta$$

$$\lambda_i \in [0, 1[\,,\ n_i \in \mathbb{Z}$$

impose behaviour of Expansion by Regions → impose **cancellation** of **unwanted singularities** → linear relations between coefficients

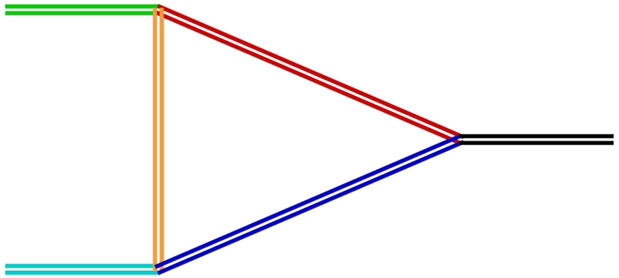
only 1-loop tadpole and **massless bubble** needed for **32 MIs**

$\forall\, \varepsilon$



- Method
- Implementation
- **Examples**

1-Loop Triangle with Internal and External Masses



input card written for a specific run

```
order: 5
precision: 16
point: [
  p12 = 1,
  p22 = 2,
  s = 3,
  m12 = 100,
  m22 = 100,
  m32 = 100
]
exit-sing: 1
gen-bound: 1
```

order → epsilon orders

precision → precision digits

point → target point

exit-sing → start from designated singular point

gen-bound → automated BC generation

common files written once per topology

```
common/
├── vars.txt
├── 0.txt
├── 1.txt
├── 2.txt
├── 3.txt
├── 4.txt
├── 5.txt
├── branch_cuts.txt
├── initial_point.txt
├── bound_behav.txt
└── bound_build.txt
```

vars.txt → list of variables: $p_1^2, p_2^2, s, m_1^2, m_2^2, m_3^2$

0.txt – 5.txt → DE matrices (one file per variable)

branch_cuts.txt → list of branch cuts

```
tot-branch: 3
massless-branch: 0
branches: [
  s-(m1+m3)^2,
  p12-(m1+m2)^2,
  p22-(m2+m3)^2
]
```

initial_point.txt → singular point where automated BCs are generated →

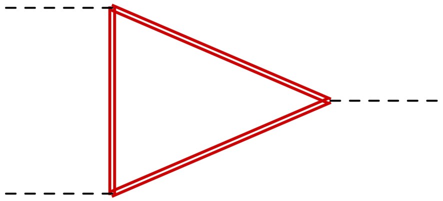
```
point: [
  s = 0,
  p12 = 0,
  p22 = 0,
  m12 = 100,
  m22 = 100,
  m32 = 100
]
```

bound_behav.txt → Expansion by Region exponents (all MIs are regular)

bound_build.txt → first three MIs are tadpoles

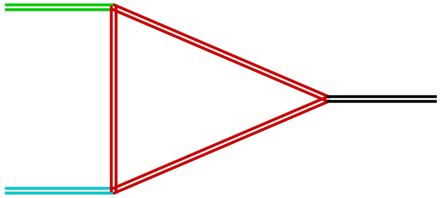
1-Loop Triangle with Internal and External Masses

```
point: [
  s = 0,
  p12 = 0,
  p22 = 0,
  m12 = 100,
  m22 = 100,
  m32 = 100
]
```



exit sing

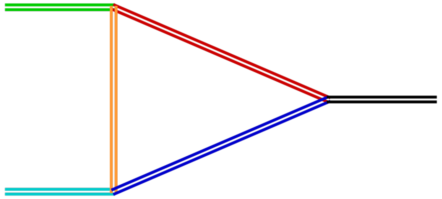
```
order: 5
precision: 16
point: [
  p12 = 1,
  p22 = 2,
  s = 3,
  m12 = 100,
  m22 = 100,
  m32 = 100
]
exit-sing: 1
gen-bound: 1
```



propagation *(58 points, 4 singular)*

```
PATH:
58 points
0. tag: 1,  eta = (0 0)
1. tag: 2,  eta = (3.36572323465479328330e-1 0)
  regular propagations...
12. tag: 2, eta = (6.72980304976141474604e-1 0)
13. tag: 1, eta = (6.73005997310859194193e-1 0)
14. tag: 0, eta = (6.73144646930958656660e-1 0)
15. tag: 1, eta = (6.73283296551058119128e-1 0)
16. tag: 0, eta = (6.73421946171157581595e-1 0)
17. tag: 1, eta = (6.73560595791257044062e-1 0)
18. tag: 2, eta = (6.73629920601306775296e-1 0)
  regular propagations...
36. tag: 2, eta = (7.47458256029787437846e-1 0)
37. tag: 1, eta = (7.47811275762189753280e-1 0)
38. tag: 0, eta = (7.48540850508126465179e-1 0)
39. tag: 1, eta = (7.49270425254063177078e-1 0)
40. tag: 0, eta = (7.50000000000000000000e-1 0)
41. tag: 1, eta = (7.50729574745936822921e-1 0)
43. tag: 2, eta = (7.51641543178357851573e-1 0)
  regular propagations...
56. tag: 2, eta = (9.99647230748999970906e-1 0)
57. tag: 1, eta = (1.00000000000000000000e0 0)
```

```
order: 5
precision: 16
point: [
  s = -1,
  m12 = 2,
  m22 = 3,
  m32 = 5,
  p12 = 1/2,
  p22 = 1/3
]
starting-point: [
  p12 = 1,
  p22 = 2,
  s = 3,
  m12 = 100,
  m22 = 100,
  m32 = 100
]
exit-sing: 0
gen-bound: 0
```



1-Loop Triangle with Internal and External Masses

MIs computed in $\varepsilon = \frac{101}{146700}$

```
MI0: (2.90441164026269689878103960...e3 0)
MI1: (4.35540146043925901671524332...e3 0)
MI2: (7.25644994136370315072624217...e3 0)
MI3: (1.45102408899755166053045532...e3 0)
MI4: (1.45063139101440792324071813...e3 0)
MI5: (1.45053840930241730004028775...e3 0)
MI6: (−1.54995337255856547712424l9...e−1 0)
```

MIs computed in $\varepsilon = \frac{17}{24450}$

```
MI0: (2.87593175196824945061702911...e3 0)
MI1: (4.31268163190383444307933535...e3 0)
MI2: (7.18525024330887846376219687...e3 0)
MI3: (1.43678414841565105694008735...e3 0)
MI4: (1.4363914550165960790255637...e3 0)
MI5: (1.4362984743558711152632678...e3 0)
MI6: (−1.54993512768638306135466447...e−1 0)
```

⋮

interpolation →

bubble

```
MI5
eps^−2: (0 0)
eps^−1: (9.9999999999999999999999e−1 0)
eps^0:  (−1.938704283006374112859e0 0)
eps^1:  (2.712629538677921993042e0 0)
eps^2:  (−3.230718213168225851725e0 0)
eps^3:  (3.549244487845649233171e0 0)
eps^4:  (−3.732208990036951965128e0 0)
eps^5:  (3.832940123529834150758e0 0)
```

triangle

```
MI6
eps^−2: (0 0)
eps^−1: (0 0)
eps^0:  (−1.551797836179781000156e−1 0)
eps^1:  (2.681530117390114836628e−1 0)
eps^2:  (−3.622993677685175397293e−1 0)
eps^3:  (4.213393862653823166643e−1 0)
eps^4:  (−4.570669082193926592410e−1 0)
eps^5:  (4.771475949009888358828e−1 0)
```

✓

AMFlow (Mathematica), triangle:

$$-0.1551797836179781 + 0.2681530117390115\,\text{eps} - 0.3622993677685175\,\text{eps}^2 + 0.4213393862653823\,\text{eps}^3 - 0.4570669082193927\,\text{eps}^4 + 0.4771475949009888\,\text{eps}^5$$

1-Loop Triangle with Internal and External Masses

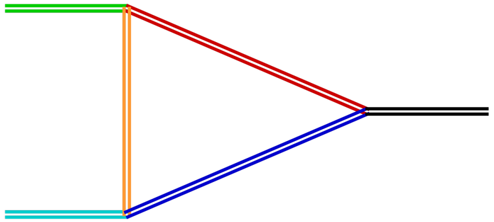
```
point: [
  s = −1,
  m12 = 2,
  m22 = 3,
  m32 = 5,
  p12 = 1/2,
  p22 = 1/3
]
```

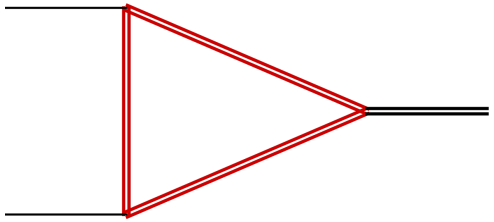
propagation *(8 points, 3 singular)*

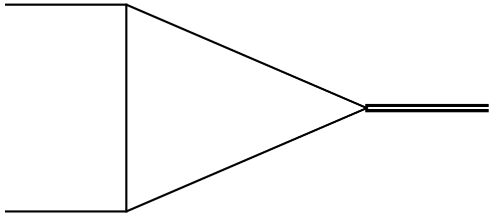
```
point: [
  s = −1,
  m12 = 2,
  m22 = 3,
  m32 = 5,
  p12 = 0,
  p22 = 0
]
```

propagation *(5 points, 1 singular)*

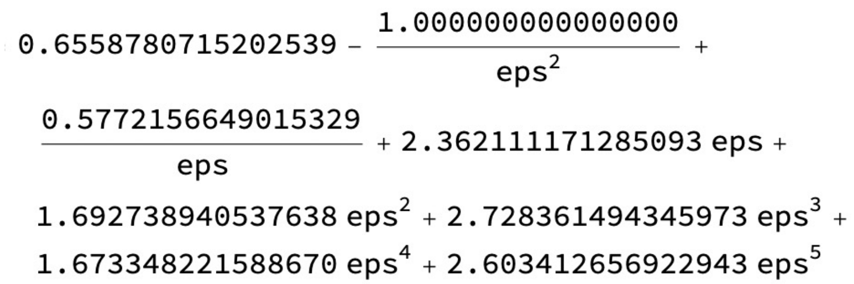
```
point: [
  s = −1,
  m12 = 0,
  m22 = 0,
  m32 = 0,
  p12 = 0,
  p22 = 0
]
```







AMFlow (Mathematica):



$$0.6558780715202539 - \frac{1.000000000000000}{\text{eps}^2} + \frac{0.5772156649015329}{\text{eps}} + 2.362111171285093\,\text{eps} + 1.692738940537638\,\text{eps}^2 + 2.728361494345973\,\text{eps}^3 + 1.673348221588670\,\text{eps}^4 + 2.603412656922943\,\text{eps}^5$$

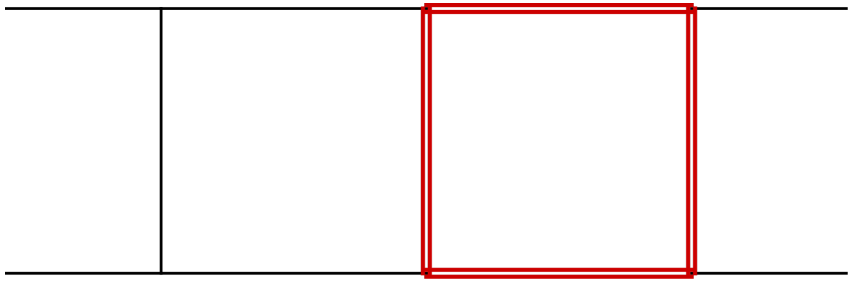
interpolated triangle:

```
order: 5
precision: 16
```

```
MI6
eps^−2: (−9.999999999999999999999999999999999999748437e−1 0)
eps^−1: (5.7721566490153286060651209008240243104215882264537 17e−1 0)
eps^0:  (6.5587807152025388107701951514539048128465406222794 33e−1 0)
eps^1:  (2.3621111712850933352704723881474701340107082855250 22e0 0)
eps^2:  (1.6927389405376376611765313232967988418809451764631 19e0 0)
eps^3:  (2.7283614943459733436498867162976484775549232116739 21e0 0)
eps^4:  (1.6733482215886704514582969317806961527054556661735 32e0 0)
eps^5:  (2.6034126569229428838390683542444793481172828739478 41e0 0)
```

✓

2-Loop Box with a Massive Loop



input card written for a specific run

```
order: 5
precision: 16
point: [
  s = 1,
  t = 2,
  m2 = 100
]
exit-sing: 1
gen-bound: 1
```

- order → epsilon orders
- precision → precision digits
- point → target point
- exit-sing → start from designated singular point
- gen-bound → automated BC generation

common files written once per topology

```
common/
├── vars.txt
├── 0.txt
├── 1.txt
├── 2.txt
├── branch_cuts.txt
├── initial_point.txt
├── bound_behav.txt
└── bound_build.txt
```

- vars.txt → list of variables: s, t, m^2
- 0.txt, 1.txt, 2.txt → DE matrices
- branch_cuts.txt → list of branch cuts

```
tot-branch: 3
massless-branch: 1
branches: [
  s,
  s-4*m^2,
  t-4*m^2
]
```

- initial_point.txt → singular point where automated BC are obtained

```
point: [
  s = 0,
  t = 0,
  m2 = 100
]
```

- bound_behav.txt → Expansion by Region exponents
- bound_build.txt →
  - 1^{st} MI: tadpole squared
  - 2^{nd} MI: tadpole times massless bubble

2-Loop Box with a Massive Loop

```
point: [
  s = 0,
  t = 0,
  m2 = 100
]
```

exit sing

automated BCs

```
point: [
  s = 1,
  t = 2,
  m2 = 100
]
exit-sing: 1
gen-bound: 1
```

propagation *(62 points, 6 singular)*

```
point: [
  s = -63845/42,
  t = 1000/11,
  m2 = 100
]
```

AMFlow
(Mathematica):

```
order: 16
precision: 32
```

```
MI n.28
j[box1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
-0.12242015136700982584490132491802
MI n.29
j[box1, 1, 1, 1, 1, 1, 1, 1, -1, 0]
24.549416261115754491861165077516
MI n.30
j[box1, 1, 1, 1, 1, 1, 1, 1, 0, -1]
0.024111291491943627972044180058385
MI n.31
j[box1, 1, 1, 1, 1, 1, 1, 1, -2, 0]
-6.3619783877291987620952972887149 × 10^7
```

MIs computed in $\varepsilon = \frac{101}{464100}$

```
MI28: (-1.22420151367009825844901324918016631З...e-1 0)
```

```
MI28: (-1.224201513670098258449013249180166313...e-1 0)
       -0.12242015136700982584490132491802
MI29: (2.454941626111575449186116507751561747 4...e1 0)
        24.549416261115754491861165077516
MI30: (2.411129149194362797204418005838490138 0...e-2 0)
      0.024111291491943627972044180058385
MI31: (-6.361978387729198762095297288714888052...e7 0)
         6.3619783877291987620952972887149*10^7
```

✓

2-Loop Box with a Massive Loop

```
point: [
  s = 0,
  t = 0,
  m2 = 100
]
```

exit sing

automated BCs

```
point: [
  s = 1,
  t = 2,
  m2 = 100
]
exit-sing: 1
gen-bound: 1
```

***propagation** (62 points, 6 singular)*

```
point: [
  s = -63845/42,
  t = 1000/11,
  m2 = 100
]
```

AMFlow (Mathematica):

```
order: 16
precision: 32
```

$$-\frac{5.8115102682076014567247336890732\times10^{-9}}{\text{eps}^2}+\frac{6.2503121765501406847317590673082\times10^{-8}}{\text{eps}}+$$
$$-3.3952235038847814964141719685324\times10^{-7}+$$
$$1.2461176087695035216668922779277\times10^{-6}\,\text{eps}-$$
$$3.4821760387881041540787251794568\times10^{-6}\,\text{eps}^2+$$
$$7.9119695037241438698853459388438\times10^{-6}\,\text{eps}^3-$$
$$0.000015240088337025816357827732623299\,\text{eps}^4+$$
$$0.000025617216303446256915950005670167\,\text{eps}^5$$

```
order: 5
precision: 16
```

interpolation

```
MI28
eps^-4: (0 0)
eps^-3: (0 0)
eps^-2: (-5.8115102682076014567247336890731643 6...e-9 0)
   -5.8115102682076014567247336890732*10^-9
eps^-1: (6.25031217655014068473175906730822546...e-8 0)
   6.2503121765501406847317590673082*10^-8
eps^0: (-3.39522350388478149641417196853243216...e-7 0)
   -3.3952235038847814964141719685324*10^-7
eps^1: (1.24611760876950352166689227792774788...e-6 0)
   1.2461176087695035216668922779277*10^-6
eps^2: (-3.48217603878810415407872517945677456...e-6 0)
   -3.4821760387881041540787251794568*10^-6
eps^3: (7.911969503724143869885345938843111 09...e-6 0)
   7.9119695037241438698853459388438*10^-6
eps^4: (-1.5240088337025816357827732619384586 8...e-5 0)
  -0.000015240088337025816357827732623299
eps^5: (2.5617216303446256915949988984466244 7...e-5 0)
  0.000025617216303446256915950005670167
```

✓

2-Loop Box with a Massive Loop

```
point: [
  s = 0,
  t = 0,
  m2 = 100
]
```

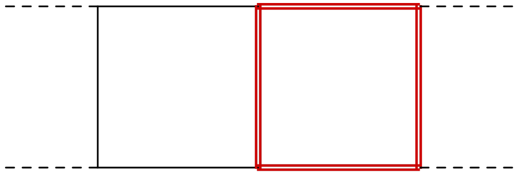
exit sing

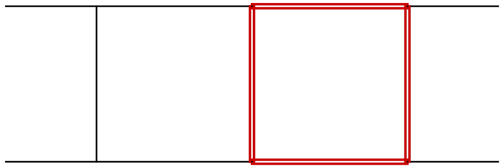
automated BCs

```
point: [
  s = 1,
  t = 2,
  m2 = 100
]
exit-sing: 1
gen-bound: 1
```

propagation *(62 points, 6 singular)*

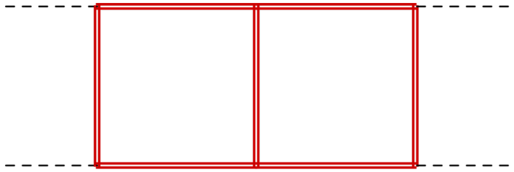
```
point: [
  s = -63845/42,
  t = 1000/11,
  m2 = 100
]
```





set mass to zero

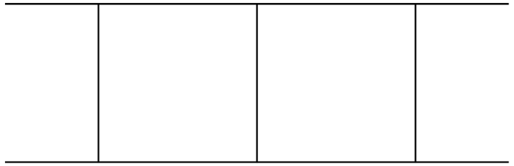
2-loop **massless** box +
auxiliary complex mass



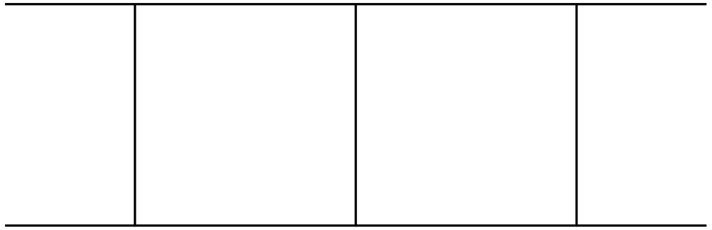
AMFlow *method*

```
point: [
  s = 1,
  t = 2,
  m2 = 0
]
```

consistency check



2-Loop Massless Box with AMFlow Method



input card:

```
order: 5
precision: 16
point: [
        s = 1,
        t = 2,
]
exit-sing: -1
```

use AMFlow method

needed for **interface to `Kira`**

```
external-momenta: [p1, p2, p3, p4,]
momentum-conservation: [p4, -p1-p2-p3]
masses: []
loop-momenta: [k1, k2,]
isp: 2
propagators: [
  [k1, 0],
  [k1-p1, 0],
  [k1+p2, 0],
  [k2-p2, 0],
  [k2+p1, 0],
  [k1+k2, 0],
  [k2-p2-p3, 0],
  [k2, 0 ],
  [k1 + p3, 0]
]
kinematic-invariants: [s, t]
squared-momenta: [
  [p1, 0],
  [p2, 0],
  [p3, 0],
  [p1+p2, s],
  [p2+p3, t],
  [p1+p3, -s-t]
]
```

two additional **common files**:

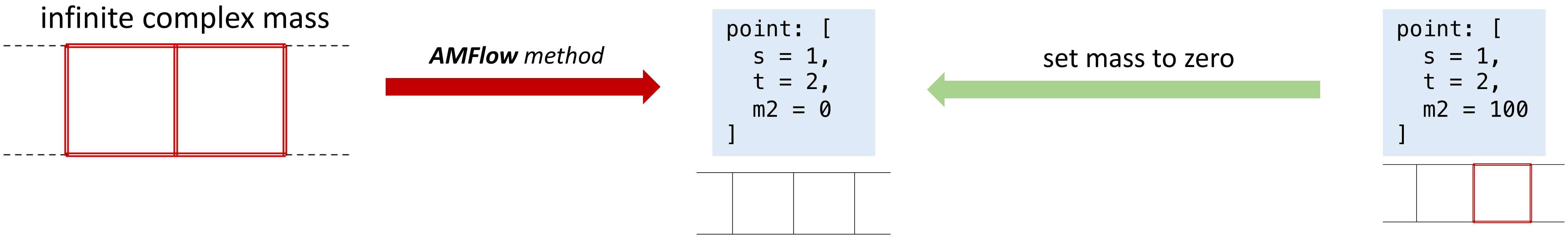
```
common/
├── ...
├── ...
├── ...
├── topology.txt
└── MIs.txt
```

list of master integrals
(for eta-less propagation)

```
MI[0, 1, 0, 1, 0, 1, 0, 0, 0]
MI[1, 0, 0, 0, 0, 1, 1, 0, 0]
MI[0, 1, 1, 1, 1, 0, 0, 0, 0]
MI[1, 0, 0, 1, 1, 1, 0, 0, 0]
MI[1, 1, 1, 0, 0, 1, 1, 0, 0]
MI[1, 1, 0, 1, 0, 1, 1, 0, 0]
MI[1, 1, 1, 1, 1, 1, 1, -1, 0]
MI[1, 1, 1, 1, 1, 1, 1, 0, 0]
```

(or just one target integral)

2-Loop Box with a Massive Loop



infinite complex mass

***AMFlow** method*

```
point: [
  s = 1,
  t = 2,
  m2 = 0
]
```

set mass to zero

```
point: [
  s = 1,
  t = 2,
  m2 = 100
]
```

```
order: 5
precision: 16
```

```
MI38 (MI7 eta-less)
eps^-4: ( 1.999999999999999999999999999999...e0   0)
eps^-3: (-4.041730611005994715969128663975051...e0   1.256637061435917295385057353311 80...e1)
eps^-2: (-4.800178134306239820648797501016090...e1  -2.539494239065083864940852074135 63...e1)
eps^-1: ( 8.498679680667727221592808504750474...e1  -1.362372782915778166568532524881 58...e2)
eps^0:  ( 3.260144789203066943599377257841968...e2   1.998037460877339885470307139625 84...e2)
eps^1:  (-3.501310096881496451391002426392785...e2   6.908266581269574526736784496489 55...e2)
eps^2:  (-1.305709966174408174011009837021440...e3  -4.501632640527147386171776855657 70...e2)
eps^3:  ( 4.648584443413964371500437987977635...e2  -2.195200257214888078247287468389 53...e3)
eps^4:  ( 4.183260324015061098269632785110026...e3   6.100250465980061070325139148179 31...e2)
eps^5:  ( 2.497386154530619817602981775742910...e3   1.004225908131303403892868179111 06...e4)
```

✓

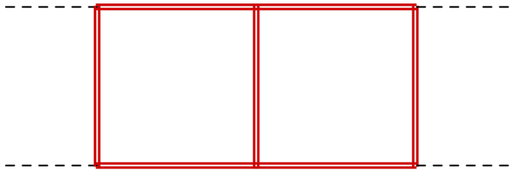
AMFlow (Mathematica):

```
order: 16
precision: 32
```

$$\frac{(326.01447892030669435993772578420 + 199.80374608773398854703071396258\,i) + 2.0000000000000000000000000000000}{eps^4} - \frac{4.0417306110059947159691286639751 - 12.566370614359172953850573533118\,0\,i}{eps^3} - \frac{48.001781343062398206487975010161 + 25.394942390650838649408520741356\,i}{eps^2} + \frac{84.986796806677272215928085047 50 - 136.23727829157781665685325248816\,i}{eps} - (350.13100968814964513910024263928 - 690.82665812695745267367844964896\,i)\,eps - (1305.7099661744081740110098370214 3 + 450.16326405271473861717768556577\,i)\,eps^2 + (464.85844434139643715004379875 62 - 2195.2002572148880782472874683784\,i)\,eps^3 + (4183.2603240150610982696330065891 + 610.02504659800610703251385547 03\,i)\,eps^4 + (2497.3861545306198176020379045411 + 10\,042.259081313034038928934713331 3\,i)\,eps^5$$

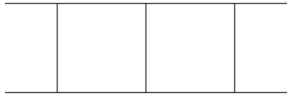
2-Loop Box with a Massive Loop

infinite complex mass



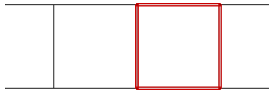
***AMFlow** method*

```
point: [
  s = 1,
  t = 2,
  m2 = 0
]
```



set mass to zero

```
point: [
  s = 1,
  t = 2,
  m2 = 100
]
```



```
MI38 (MI7 eta-less)
eps^-4: ( 1.9999999999999999999999999999999...e0   0)
eps^-3: (-4.041730611005994715969128663975051...e0   1.25663706143591729538505735331180...e1)
eps^-2: (-4.800178134306239820648797501016090...e1  -2.53949423906508386494085207413563...e1)
eps^-1: ( 8.498679680667727221592808504750474...e1  -1.36237278291577816656853252488158...e2)
eps^0:  ( 3.260144789203066943599377257841968...e2   1.99803746087733988547030713962584...e2)
eps^1:  (-3.501310096881496451391002426392785...e2   6.90826658126957452673678449648955...e2)
eps^2:  (-1.305709966174408174011009837021440...e3  -4.50163264052714738617177685565770...e2)
eps^3:  ( 4.648584443413964371500437987977635...e2  -2.19520025721488807824728746838953...e3)
eps^4:  ( 4.183260324015061098269632785110026...e3   6.10025046598006107032513914817931...e2)
eps^5:  ( 2.497386154530619817602981775742910...e3   1.00422590813130340389286817911106...e4)
```

✓

also consistent with
massive to massless propagation

✓

```
MI28
eps^-4: ( 1.9999999999999999999e0  0)
eps^-3: (-4.04173061100599471596e0  1.2566370614359172953850e1)
eps^-2: (-4.80017813430623982064e1 -2.5394942390650838649408e1)
eps^-1: ( 8.49867968066772722159e1 -1.3623727829157781665685e2)
eps^0:  ( 3.26014478920306694359e2  1.9980374608773398854703e2)
eps^1:  (-3.50131009688149645139e2  6.9082665812695745267367e2)
eps^2:  (-1.30570996617440817401e3 -4.5016326405271473861717e2)
eps^3:  ( 4.64858444341396437150e2 -2.1952002572148880782472e3)
eps^4:  ( 4.18326032401506109826e3  6.1002504659800610703251e2)
eps^5:  ( 2.49738615453061981760e3  1.0042259081313034038928e4)
```

Conclusions and Outlook

- **LINE** (**L**oop **I**ntegral **N**umerical **E**valuator) propagate **loop integrals** via **DEs** with an **only-what-is-needed** approach
- **C** implementation of light symbolical structures such as **expression trees**, **fractions of polynomial** etc.
- towards **on the fly evaluation** of loop integrals
- **interface to Kira** and implementation of the **AMFlow method**
- **automated BCs** on selected examples with massive lines (generalization under investigation)
- **open source** code available in a few weeks

Conclusions and Outlook

- **LINE** (**L**oop **I**ntegral **N**umerical **E**valuator) propagate **loop integrals** via **DEs** with an **only-what-is-needed** approach
- **C** implementation of light symbolical structures such as **expression trees**, **fractions of polynomial** etc.
- towards **on the fly evaluation** of loop integrals
- **interface to Kira** and implementation of the **AMFlow method**
- **automated BCs** on selected examples with massive lines (generalization under investigation)
- **open source** code available in a few weeks

Stay tuned!

Backup

Mathematical expressions in C

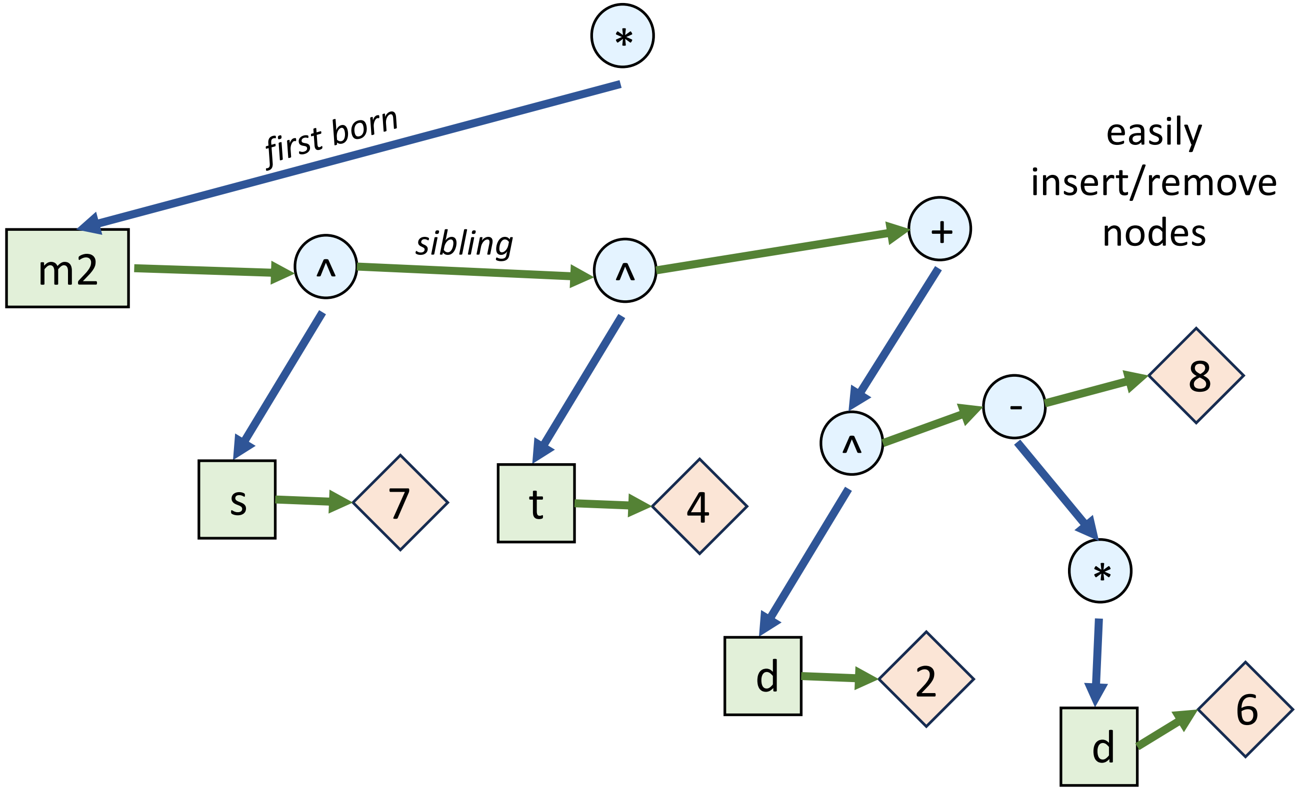
Most operations performed with **recursive algorithms**

- climb down the tree until desired bottom nodes are reached
- perform operation on all siblings
- climb back the tree

substitute phase-space line,
get polynomial coefficients,
find roots, ...

m2*s^7*t^4*(8-6*d+d^2)

string representing a
mathematical expression

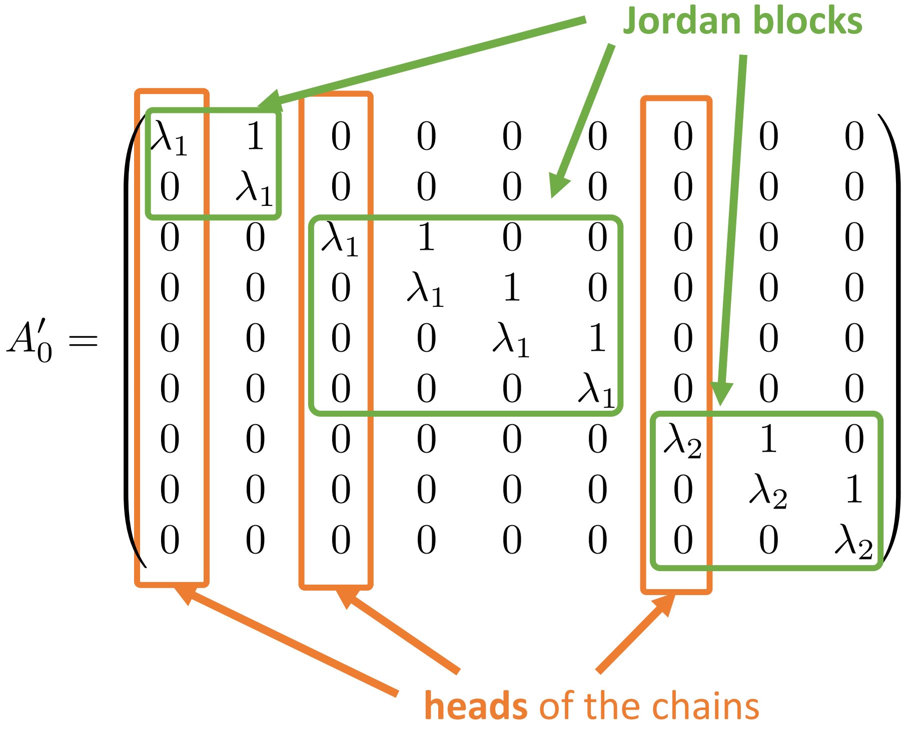


easily
insert/remove
nodes

Jordan Normal Form

Not all $N\times N$ matrices are diagonalizable → **defective matrices**: fewer than N eigenvectors

Jordan normal form:



$$A_0' = \begin{pmatrix} \lambda_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 \end{pmatrix}$$

Jordan chain:

$$A_0'\vec{v_1} = \lambda\vec{v_1}$$ head of the chain (eigenvector)

$$A_0'\vec{v_i} = \lambda\vec{v_i} + \vec{v}_{i-1} \quad i = 2,\ldots,L$$

generalized eigenvectors

defective eigenvalues: g.m. ≠ a.m.

λ_1: g.m. = 2, a.m. = 6

λ_2: g.m. = 1, a.m. = 3

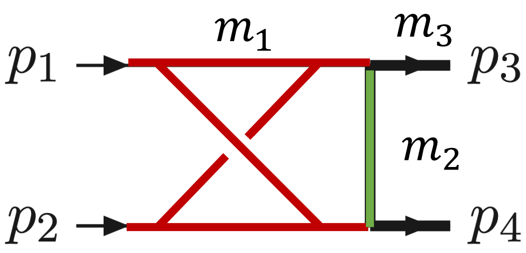
ill-defined numerical problem:
the structure of the output is affected by the **round-off error**!

$$\lambda_1 \overset{?}{=} \lambda_2$$

it depends on the precision!

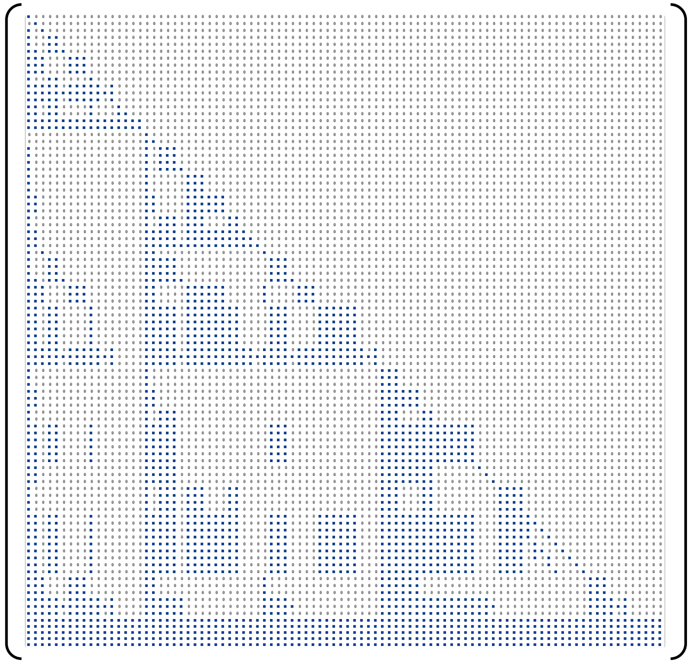
Change to Fuchsian Basis

- 2-loop non-planar double-box with additional masses → 92 MIs



(two internal varying masses,
massless initial state,
massive final state)

block structure



initial Poincaré rank = 3

correctly transformed
to **Fuchsian form** ✓

speed test:

time using **`Mathematica`**:
0.254574 sec

time using **`LINE`**:
0.00258276 sec

O(100) speed
improvement ✓