



MAX-PLANCK-INSTITUT
FÜR PHYSIK



DIPARTIMENTO
DI **FISICA**
ETTORE PANCINI



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



Feynman Integrals Numerical Evaluation with **LINE**

Renato Maria Prisco

MAX PLANCK INSTITUTE FOR PHYSICS, MUNICH

renato.prisco@mpp.mpg.de

10th International Workshop on High Precision for Hard Processes (HP2)

Karlsruhe, October 6, 2026

in collaboration with:

Jonathan Ronca (Uni. Silesia), **Francesco Tramontano** (unina & INFN)

JHEP07(2025)219

<https://github.com/line-git/line>

Part I – A quick tour of LINE

Motivations and background

The evaluation of **Feynman integrals** is one of the main bottlenecks in **higher-order calculations**. A fully numerical approach allows for automated computations when the number of loops, legs and scales increases.

COMPUTATION OF SINGLE POINT

- **pySecDec**
 - **FIESTA**
 - **AMFlow**
 - ...
- Require relatively large resources
Need nothing else
- Also a **DE solver!**
(DEs w.r.t. auxiliary mass)

COMPUTATION OF SERIES EXPANSION

- **DESS**
 - **DiffExp**
 - **SesSyde**
 - ...
- Require DEs w.r.t. kinematics
Require boundary conditions
Relatively more efficient
Can be propagated across the phase space

Solving DEs via series expansion
can serve **both purposes!**

LINE (Loop Integrals Numerical Evaluation)

A tool to compute loop integrals solving DEs via series expansion

Boundary generation and **propagation**
in a **single tool**

OPEN SOURCE

- Written in **C/C++** → enhanced performance 
- No licence needed → large cluster computations more feasible
- Code available at <https://github.com/line-git/line>

THE DE METHOD: BEYOND LOOP INTEGRALS

One can setup DEs to compute

- Integrals with delta functions → phase-space integrals
- Integrals with linear propagators → subtraction counterterms, EFT
- Special functions → Chen iterated integrals, pentagon functions

Feynman integrals via differential equations

- Get boundary conditions at one point
- Build DEs along the line connecting **boundary** and **target points**
- Solve DES via **series expansion**

$$\frac{d}{d\eta} \mathbf{I}(\eta) = \mathbb{A}(\eta) \mathbf{I}(\eta)$$

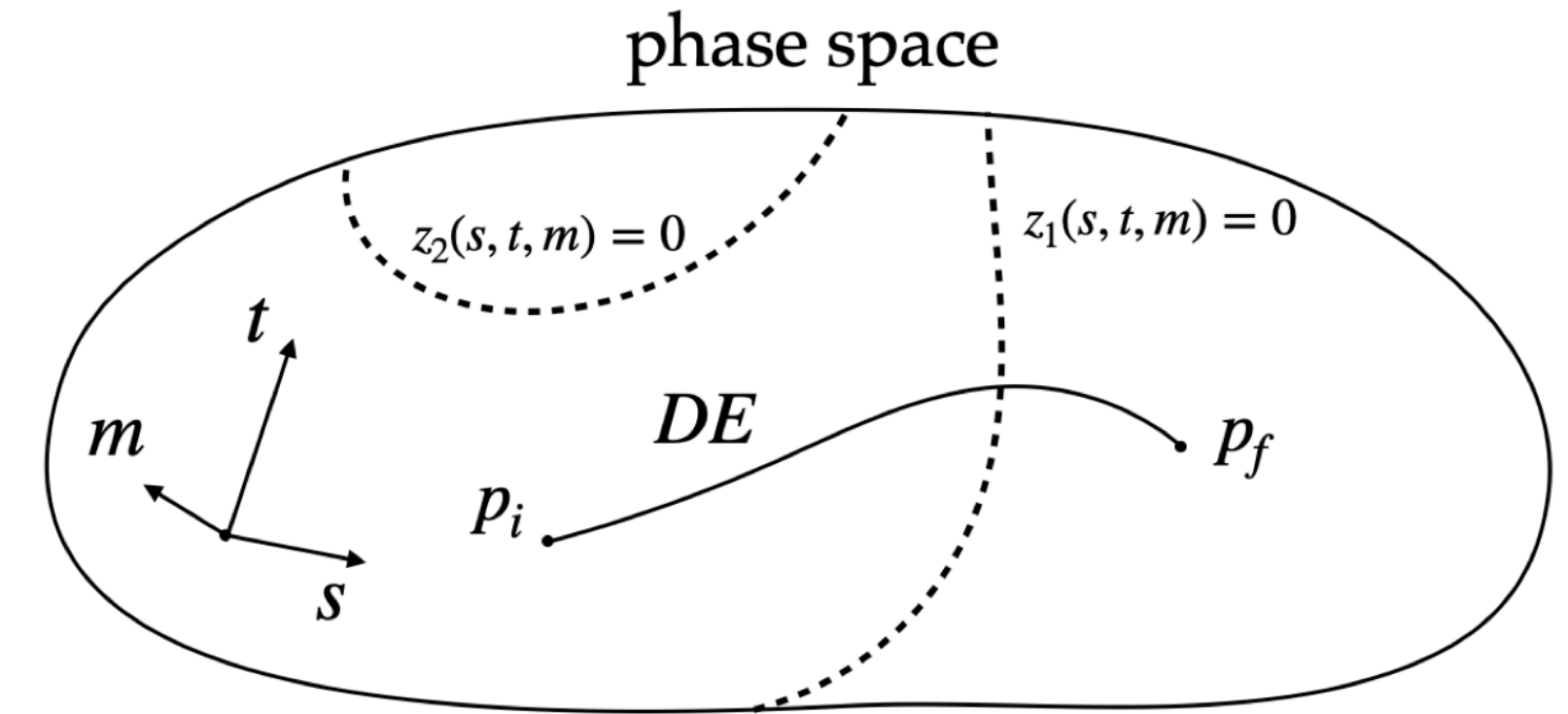
$$\begin{cases} s(\eta) = s_i + \eta(s_f - s_i) \\ t(\eta) = t_i + \eta(t_f - t_i) \\ \vdots \end{cases} \quad \eta \in [0, 1]$$

Line parameter

$$\mathbb{A}(\eta) = (s_f - s_i)\mathbb{A}_s + (t_f - t_i)\mathbb{A}_t + \dots$$

DE matrix

Set of independent **Master Integrals** (MIs)

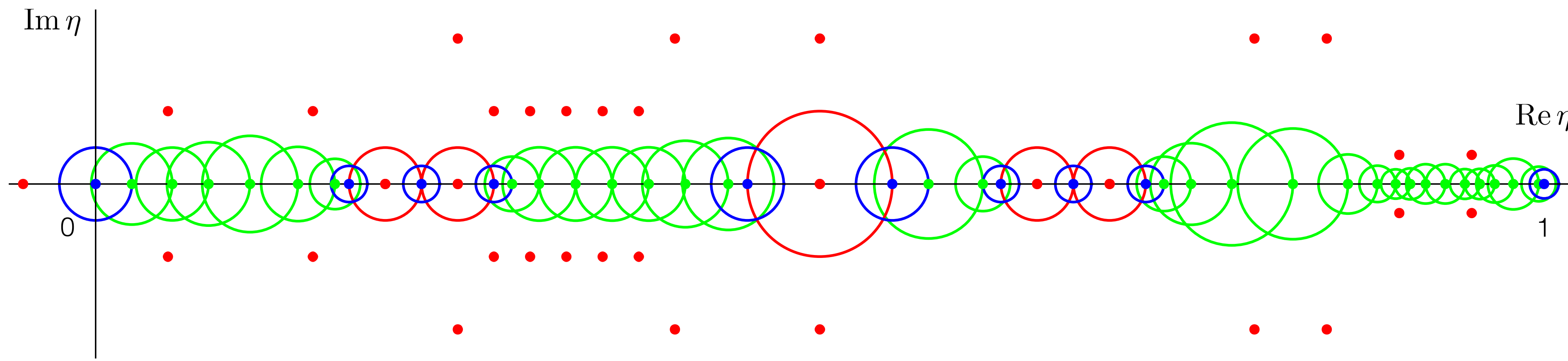


Solve for numerical values of ε and interpolate (**parallelisation**)

$$\mathbf{I}(\varepsilon, \eta) = \frac{1}{\varepsilon^p} \sum_{k=0}^{\infty} \mathbf{I}_k(\eta) \varepsilon^k$$

$$d = 4 - 2\varepsilon$$

Space-time dimension
(**dimensional regularisation**)



Series expansions have finite radius of convergence due to the presence of singularities

- poles of the DE matrix
- regular points
- matching points

Feynman integrals via differential equations

Ansatz around **regular** points

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

Ansatz around **regular-singular** points

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



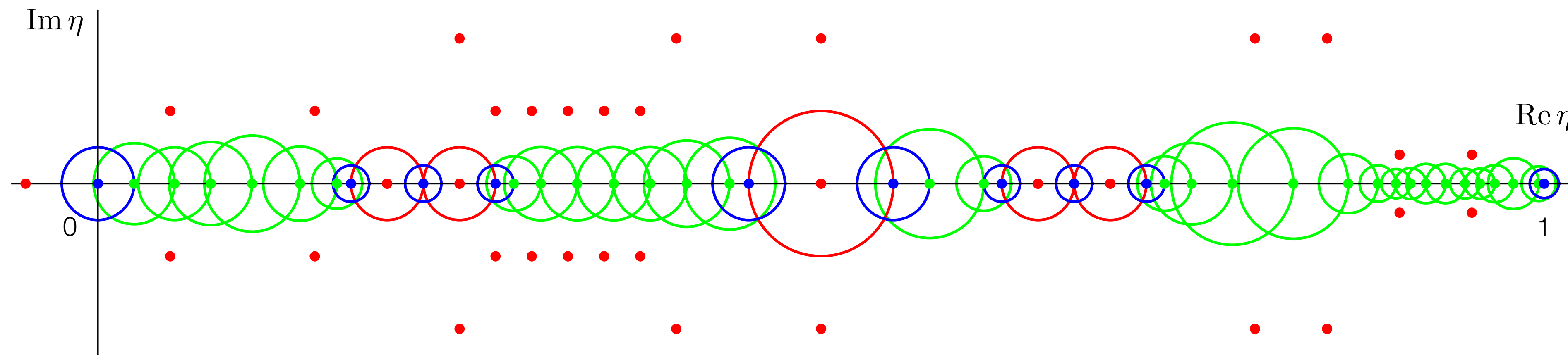
$$\frac{d}{d\eta} \mathbf{I}(\eta) = \mathbb{A}(\eta) \mathbf{I}(\eta)$$



Linear relations for $\mathbf{I}_{\lambda,l,k}$

WHAT IS NEEDED

- Parse and handle mathematical expressions
- Transform to Fuchsian form
- Boundary conditions
- Analytic continuation
- ...



Series expansions have finite radius of convergence due to the presence of singularities

- poles of the DE matrix
- regular points
- ◆ matching points

Mathematical expressions in C/C++

PARSE AND REPRESENT INPUT EXPRESSION



Input files
contain DE
matrices

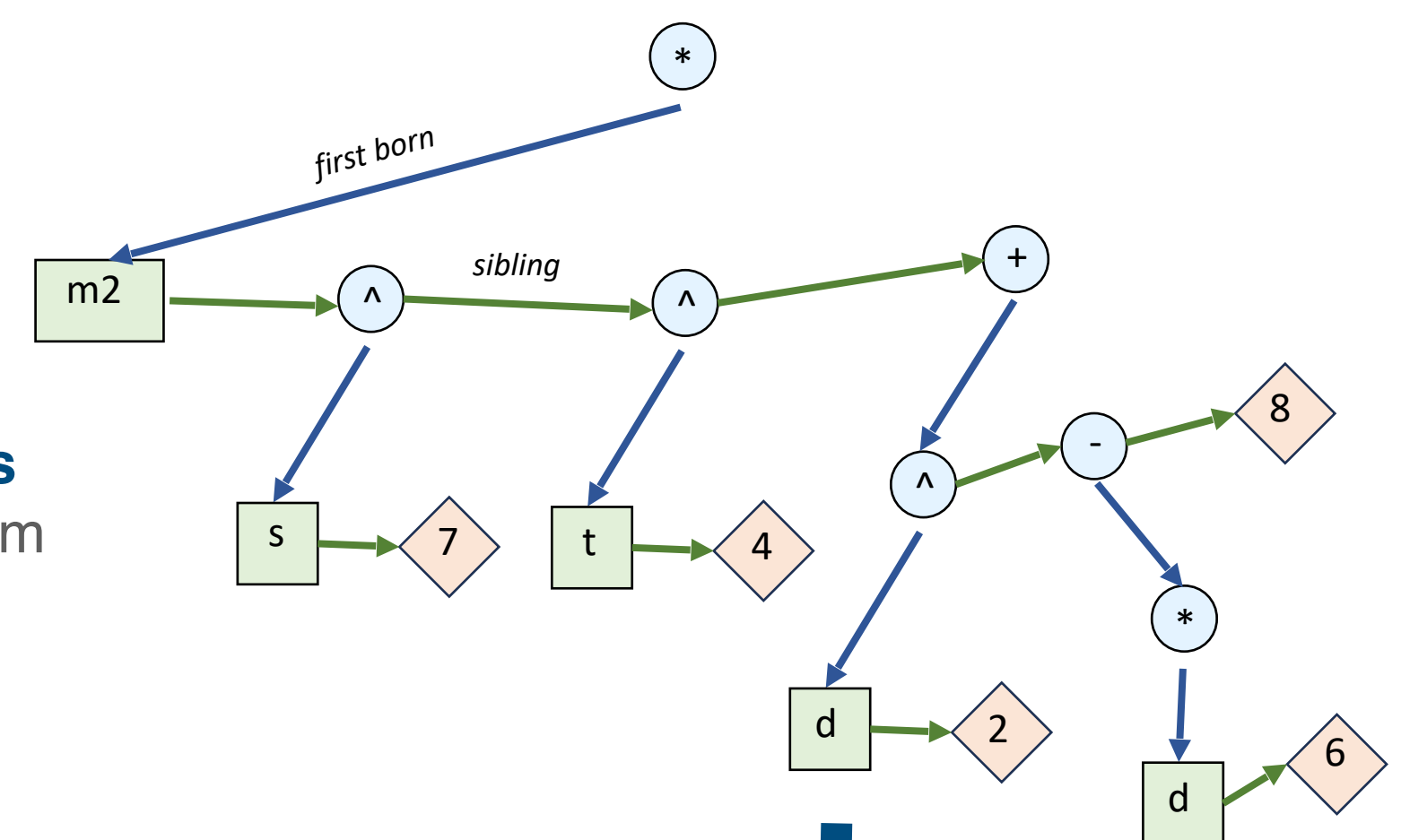
```
2*(-4+d)^2*(10-7*d+d^2)*s^9*t^5*(3*s+4*t)*(3*
(-3+d)*s+(-16+5*d)*t)-m2*s^7*t^4*(8-6*d+d^2)
(2*(-4150+3205*d-805*d^2+60*d^3)*s^4+
(-42520+32724*d-8219*d^2+675*d^3)*s^3*t+(-3
5500+27140*d-6743*d^2+532*d^3+3*d^4)*s^2*t^2+
4*(-9914+7471*d-1782*d^2+117*d^3+4*d^4)*s*t^3
+8*(-50-149*d+148*d^2-43*d^3+4*d^4)*t^4)-361
44*(-56+58*d-19*d^2+2*d^3)*m2^10*(s+t)*(4*d^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^2-19
42*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*(24
9308-242356*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^3-169
5*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+...
```

Part of a single matrix element

Mathematical expression

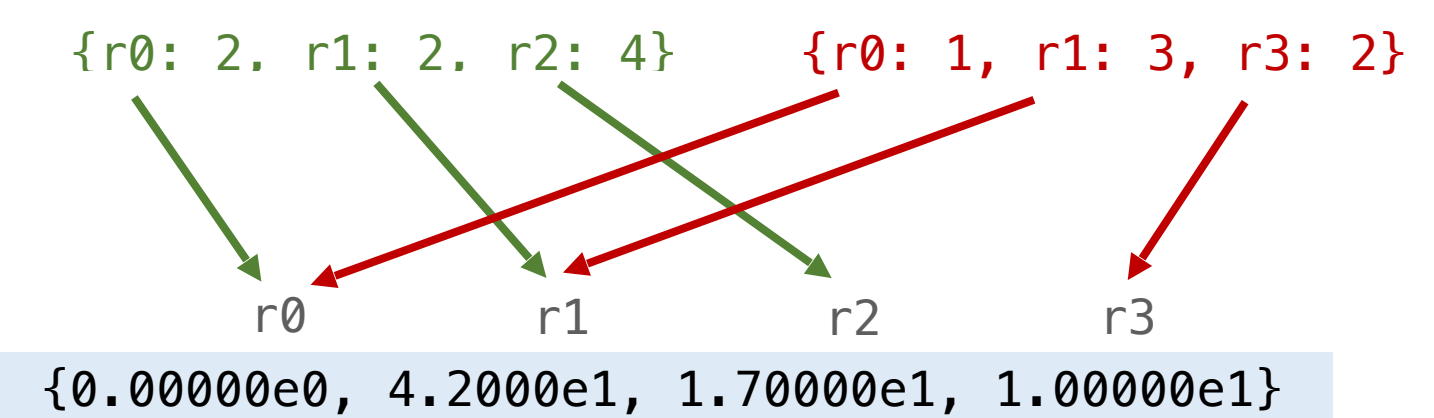
$m2*s^7*t^4*(8-6*d+d^2)$

Expression trees through linked lists
Easily insert/remove nodes and perform
operations on **sub-trees**



HANDLE RATIONAL FUNCTIONS

- Store numerator as **coefficients**
- Store denominator as **roots** and **multiplicities**
 - Scan the DE matrix and find the list of **unique denominator roots**
 - Represent denominators with **integer labels** and **multiplicities**



Global list of **unique roots**
Stored only once and accessed only when necessary

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

$$\frac{N(\eta)}{D(\eta)} = \frac{a_0 + a_1\eta + a_2\eta^2 + \dots}{\eta^{m_0}(\eta - \eta_1)^{m_1}(\eta - \eta_2)^{m_2} \dots}$$

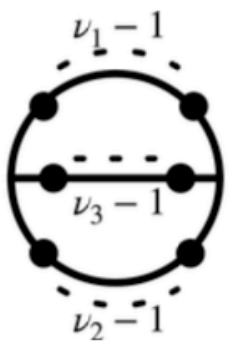
for arbitrary-precision arithmetic
N Q R C
GMP, MPFR, MPC

Boundary conditions

AMFLOW METHOD

- Introduce **auxiliary mass** (# of MIs increases)
- Get BCs at infinity (vacuum integrals)
- Propagate to zero mass

Implemented in **LINE** with interface to **Kira**
(**Kira** → IBPs → DEs. w.r.t. auxiliary mass)

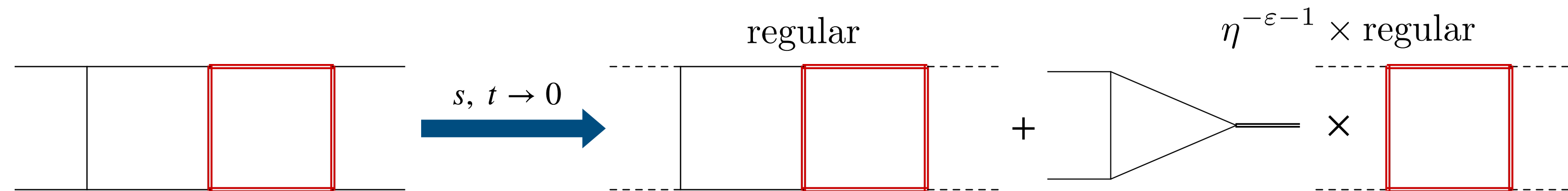


$$= (-1)^\nu \left[\frac{\Gamma(\nu_3 - 2 + \epsilon) \Gamma(\nu_1 + \nu_2 - 2 + \epsilon)}{\Gamma(\nu_3) \Gamma(\nu_1 + \nu_2)} {}_4F_3 \left(\begin{matrix} 2 - \epsilon, \nu_1, \nu_2, \nu_1 + \nu_2 - 2 + \epsilon \\ \frac{\nu_1 + \nu_2}{2}, \frac{\nu_1 + \nu_2}{2} + \frac{1}{2}, 3 - \nu_3 - \epsilon \end{matrix}; \frac{1}{4} \right) \right. \\ \left. + \frac{\Gamma(2 - \nu_3 - \epsilon) \Gamma(\nu_1 + \nu_3 - 2 + \epsilon) \Gamma(\nu_2 + \nu_3 - 2 + \epsilon) \Gamma(\nu + 2\epsilon - 4)}{\Gamma(\nu_1) \Gamma(\nu_2) \Gamma(2 - \epsilon) \Gamma(\nu + \nu_3 - 4 + 2\epsilon)} \right. \\ \left. \times {}_4F_3 \left(\begin{matrix} \nu_3, \nu_1 + \nu_3 - 2 + \epsilon, \nu_2 + \nu_3 - 2 + \epsilon, \nu - 4 + 2\epsilon \\ \nu_3 - 1 + \epsilon, \frac{\nu + \nu_3 - 4}{2} + \epsilon, \frac{\nu + \nu_3 - 3}{2} + \epsilon \end{matrix}; \frac{1}{4} \right) \right],$$

Up to **two loops**
(higher loops planned)

EXPANSION BY REGIONS (EBR)

Constraint the solution to behave around singularities as predicted by EBR (no additional parameters)



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

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

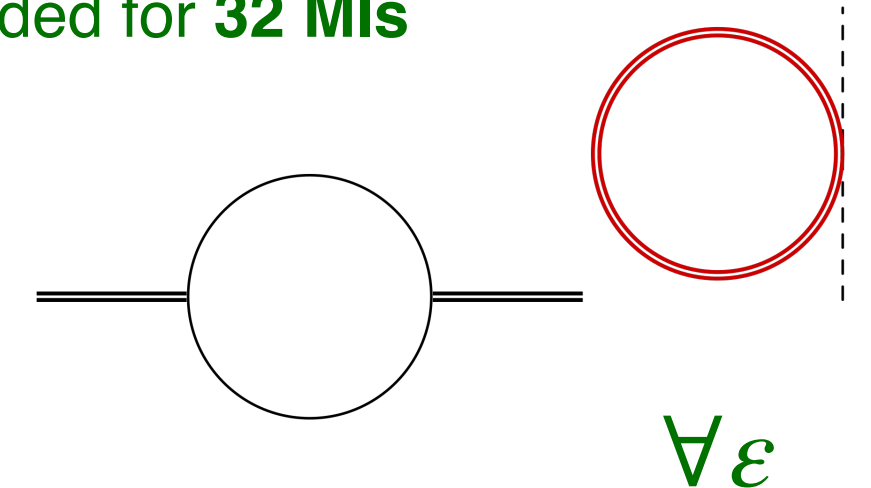
$s, t \propto \eta$

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

Compare with EBR exponents

Impose **cancellation** of
power behaviours
excluded by EBR

Linear relations
among coefficients



Successful on few examples

Solving around a singular point

FUCHSIAN TRANSFORMATION

The DEs can be written in a different basis $\eta = 0$ singular point

$$\begin{aligned}\tilde{\mathbf{I}}(\eta) &= \mathbb{T}^{-1}(\eta) \mathbf{I}(\eta) \\ \tilde{\mathbf{A}}(\eta) &= \mathbb{T}^{-1}(\eta) \mathbf{A}(\eta) \mathbb{T}(\eta) - \mathbb{T}^{-1}(\eta) \frac{d}{d\eta} \mathbb{T}(\eta)\end{aligned}$$

Change of basis

Transformation matrix

$$\frac{d}{d\eta} \mathbf{I}(\eta) = \mathbf{A}(\eta) \mathbf{I}(\eta) \longrightarrow \frac{d}{d\eta} \tilde{\mathbf{I}}(\eta) = \tilde{\mathbf{A}}(\eta) \tilde{\mathbf{I}}(\eta)$$

DE in the new basis

Building $\mathbb{T}(\eta)$ with the algorithm in [Lee '15, arxiv:1411.0911]:

$$\tilde{\mathbf{A}}(\eta) = \frac{1}{\eta} \sum_{k=0}^{\infty} \tilde{\mathbf{A}}_k \eta^k$$

DE matrix in **Fuchsian form**
It only has a simple pole

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

Related to the Jordan structure of $\mathbb{A}_0$

Eigenvalues of the leading order $\mathbb{A}_0$

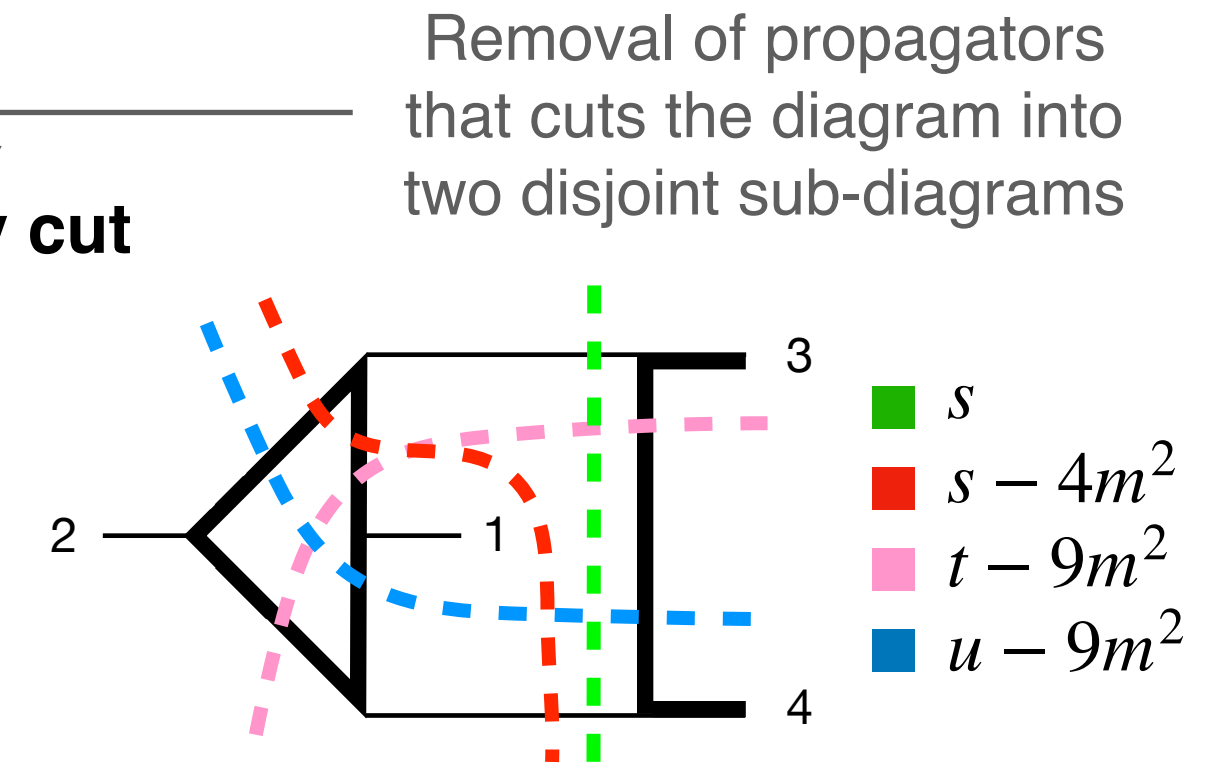
ANALYTIC CONTINUATION

Around a **branch point** the logarithms must be assigned an imaginary part in agreement with the **Feynman's prescription**

LINE prescription

Consider the associated **Cutkosky cut**

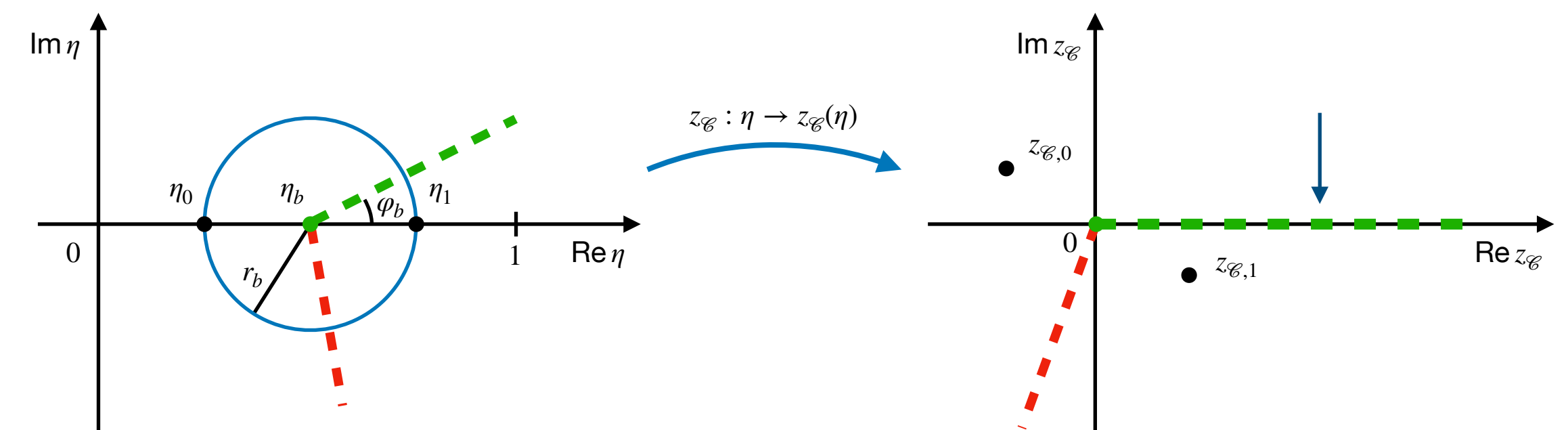
$$z_{\text{cut}} \equiv q_{\text{cut}}^2 - \left(\sum_{e \in \text{cut}} m_e \right)^2$$



- $z_{\text{cut}}(\eta) = 0$ **branch point**
- $z_{\text{cut}}(\eta) > 0$ **branch cut**

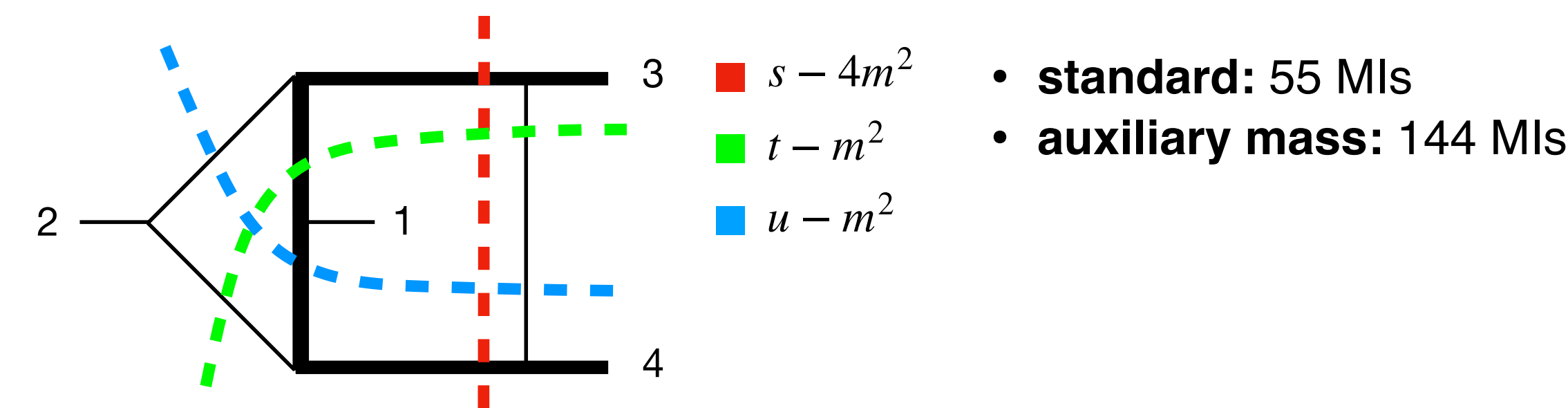
$$z_{\text{cut}} \rightarrow z_{\text{cut}} + i0^+$$

Feynman prescription
in the z_{cut} -plane



Example

2-LOOP NON-PLANAR BOX WITH A MASS



point:
s: 1
t: 2
m2: 10
exit-sing: -1
gen-bound: 1

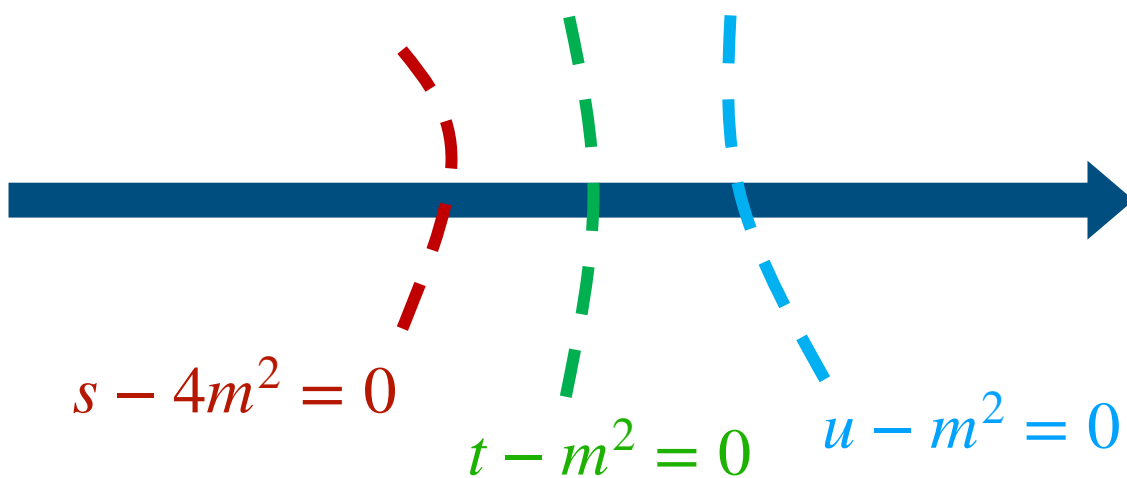
point:
s: 500
t: 150
mt2: 10
starting-point:
s: 1
t: 2
mt2: 10
exit-sing: 0
gen-bound: 0

point:
s: 500
t: 150
m2: 10
exit-sing: -1
gen-bound: 1



Q_1

19 regular and 6 singular steps



Q_2

OUTPUT

target	Q_1	Q_2
from	AMF ⁰	AMF ⁰ , Q_1 ✓ check
ϵ^{-4}	0	0
ϵ^{-3}	-2.634309928357791e-7	+7.825617108436437e-8 -2.554478084014810e-7*i
ϵ^{-2}	+2.177434402618331e-6 -1.655185743641498e-6*i	+5.136099594647812e-9 +3.245051324395477e-6*i
ϵ^{-1}	+2.177434402618331e-6 +1.533076938553119e-5*i	+5.136099594647812e-9 -3.407024087192466e-5*i
ϵ^0	-2.810879169233962e-5 -3.761642841819541e-5*i	+2.470711494037188e-4 -6.343358651146831e-5*i
ϵ^1	+6.424181660342731e-5 +3.595559671704640e-5*i	+3.561272520516187e-5 +6.872261543040661e-4*i
ϵ^2	-1.721862393547420e-4 -1.231788432398794e-5*i	-7.247299398344942e-4 +6.092012063072394e-5*i

loops: 2
order: 6
precision: 16

SCALING OF THE PERFORMANCE

precision digits				
results	decimal	8	16	32
internal	binary	313	506	893
topology	η -MIs	running time (sec)		
non-planar triangle	52	102	210	531
planar box	68	158	286	762
non-planar box	144	3066	6600	14350

AMD Opteron Processor 4386
10 cores, 64Gb RAM

$\Delta t \sim (\# \text{ digits})^{1.5}$

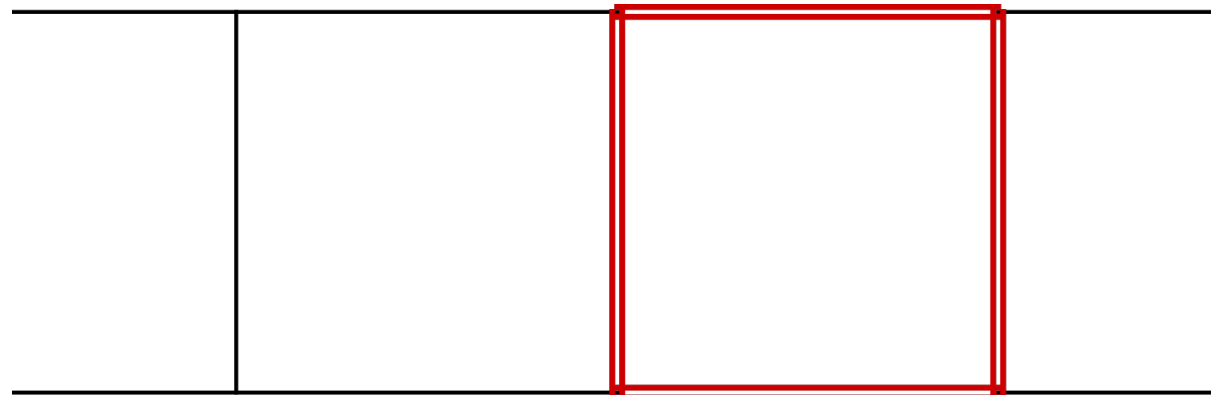
Generation of boundary values
with LINE's implementation of
the AFMlow method

Scaling dominated by numerical
rather than high-level operations

Part II – New features and improvements

YAML input interface

LINE now links **yaml-cpp** → many input files can be provided in **YAML** format



Input card written for a specific run

```
work-dir: 2L-box-m1-m
loops: 2
order: 6
precision: 16
point:
  s: 70
  t: 50
  mt2: 10
starting-point:
  s: -1
  t: 2
  mt2: 10
exit-sing: 0
gen-bound: 0
```

Common files written once per topology

```
common/
├── vars.txt    list of variables: s t m2
├── 0.txt
├── 1.txt      } DE matrices
├── 2.txt
├── branch_cuts.yaml  list of branch cuts
└── config/
    ├── kinematics.yaml
    └── integralfamilies.yaml } Kira / Reduze-like convention
```

```
kinematics:
  incoming_momenta: [p1, p2, p3, p4]
  outgoing_momenta: []
  momentum_conservation: [p4, -p1-p2-p3]
  kinematic_invariants:
  scalarproduct_rules:
    - [[p1, p1], 0]
    - [[p2, p2], 0]
    - [[p3, p3], 0]
    - [[p1+p2, p1+p2], s]
    - [[p2+p3, p2+p3], t]
    - [[p1+p3, p1+p3], -s-t]
```

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

Topology only needed for

- automated generation of branch cuts
- AMFlow method

```
integralfamilies:
  - name: MI
    loop_momenta: [k1, k2]
    top_level_sectors: [127]
    propagators:
      - [k1, 0]
      - [k1-p1, 0]
      - [k1+p2, 0]
      - [k2-p2, m2]
      - [k2+p1, m2]
      - [k1+k2, m2]
      - [k2-p2-p3, m2]
      - [k2, 0]
      - [k1 + p3, 0]
```

Simplified usage from command line

LINE can be easily run from the command line

```
$ line -i input-card.yaml
```

Run in parallel with 8 threads

Disable progress bar

Copy results in a custom file

Print results in mathematica format

Several options are available

```
$ line -i input-card.yaml -n 8 -b 0 -r results.m --out-format mathematica -w 1
```

See documentation for the full list:

- `--find-starting-pt`
- `--find-branch-cuts`
- `--singularity-detour`
- `--checkpoint`
- `--kira-parallel`
- `--incr-prec`
- `--check-prec`
- `--check-conv`
- ... and many more!

They can be collected into a file `line_args.yaml`

```
$ line -i input-card.yaml -r results.m
```

line_args.yaml

```
nthreads: 8
bar: 0
write: 1
out-format: mathematica
```

or in any other **custom file**

Write results to cache
(can be re-used as boundary for new propagations)

```
$ line -i input-card.yaml --args-file other_args.yaml -b 0
```

other_args.yaml

```
nthreads: 2
bar: 1
write: 0
result-file: debug.res
out-format: line
```

Terminal options has higher priority:
"`-b 0`" wins

Options can still be quickly overwritten
from command line

Running LINE in different modes

BOUNDARY WITH AMFLOW

```
$ line -i input-card.yaml -w 1
```

input-card.yaml

```
point:
  s: 1
  t: -2
exit-sing: -1
gen-bound: 1
```

common/

- └─ MIs.txt No need for DEs
- └─ config/
- └─ kinematics.yaml
- └─ integralfamilies.yaml

MIs.txt

```
2L-box[1, 1, 1, 1, 1, 1, 1, 0, 0]
```

Full basis of MIs or just the target integrals that the user is interested in

PHASE-SPACE PROPAGATION

```
$ line -i input-card.yaml
```

input-card.yaml

```
point:
  s: 17
  t: -42
starting-point:
  s: 1
  t: -2
exit-sing: 0
gen-bound: 0
```

common/

- └─ vars.txt
- └─ 0.txt
- └─ 1.txt
- └─ 2.txt

By default the boundary is loaded from cache

`--bound-dir path/to/bound/` to manually provide boundary values in input

DE SOLVER

```
$ line -i input-card.yaml --eps-less
```

Just 0.txt needed for univariate DE

BOUNDARY WITH EBR (EXPERIMENTAL)

```
$ line -i input-card.yaml -w 1
```

input-card.yaml

```
point:
  s: 1
  t: -2
exit-sing: 1
gen-bound: 1
```

common/

- └─ vars.txt
- └─ 0.txt
- └─ 1.txt
- └─ 2.txt
- └─ bound_behav.txt
- └─ bound_build.txt

Results of EBR analysis must be provided in input

Automated generation of branch cuts

LINE can now **automatically generate** the list of **branch cuts** for the input topology

```
$ line --find-branch-cuts 1
```



common/
└─ branch_cuts.yaml

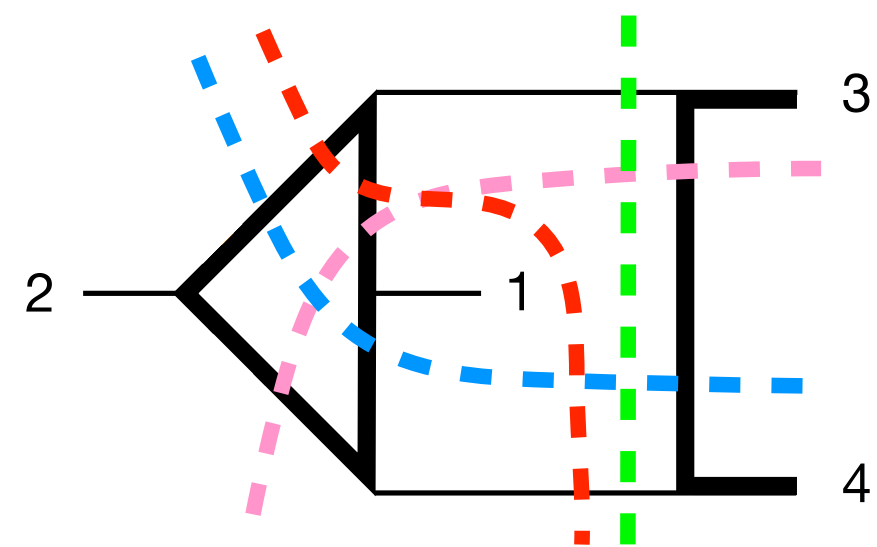
OPTION `--find-branch-cuts`

-1 (default) generate only if `branch_cuts.yaml` is absent

0 skip generation

1 regenerate and overwrite `branch_cuts.yaml` if present

2 regenerate, overwrite and stop



`branch_cuts.yaml`

`tot-branch: 4`

`massless-branch: 1`

`branches:`

■ `s`

■ `s-4*m^2`

■ `t-9*m^2`

■ `-s-t-7*m^2`

PIPELINE

Extract topology information from input files

`config/`

└─ `kinematics.yaml`

└─ `integralfamilies.yaml`

For each propagator, build a row with momentum coefficients

$$q_i = \sum_{j \in \text{loop}} a_{ij} k_j + \sum_{k \in \text{external}} b_{ik} p_k$$

$(a_{i1}, a_{i2}, \dots | b_{i1}, b_{i2}, \dots)$
loop momenta external momenta

Gauss elimination to find **minimal sets** of propagator whose loop components are linearly dependent

Get total momentum flowing through the cut applying the linear combination that cancel the loop components

Map internal kinematics into the user-defined one

Filter: eliminate duplicates, zero-momentum cuts, ...

Automated generation of branch cuts

EXAMPLE 1

1-LOOP BOX WITH A MASS

$$p_1^2 = p_2^2 = m_1^2 = m_2^2 = m_3^2 = 0$$

$$p_3^2 = p_4^2 = m_4^2 = m^2$$

1. {1,2}
2. {2,3}
3. {3,4}
4. {1,4}
5. {1,3}
6. {2,4}

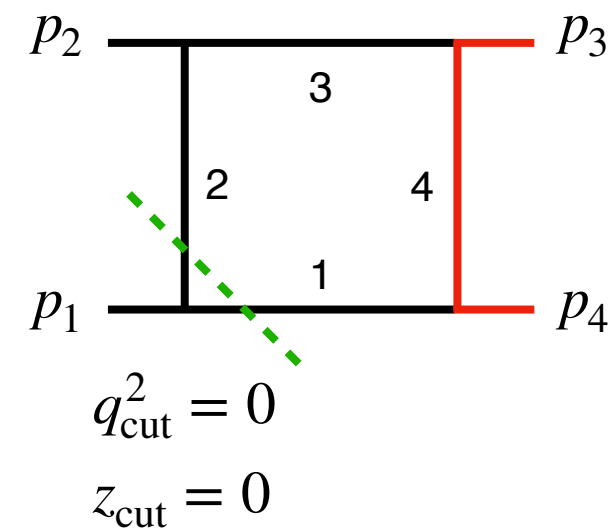
List of Cutkosky cuts

tot-branch: 2
massless-branch: 1
branches:

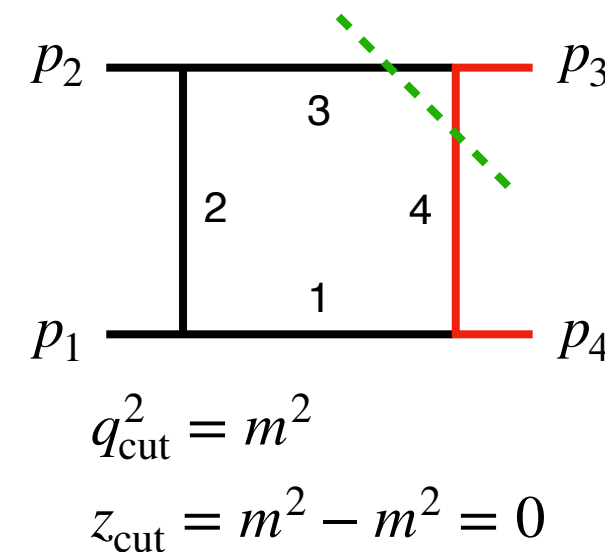
- t
- $t - m^2$

List of branch cuts

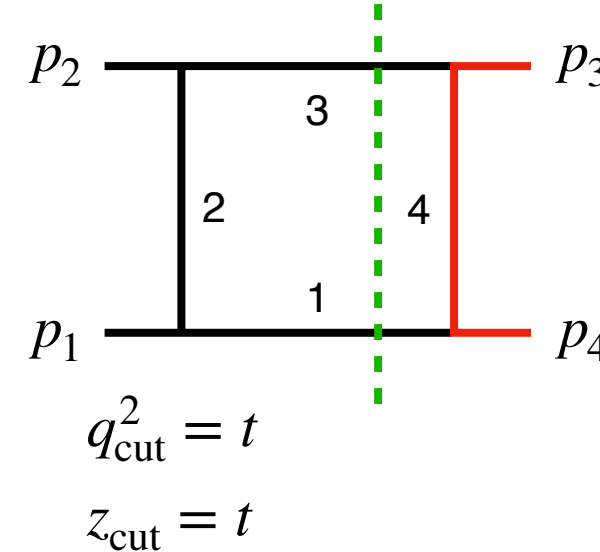
1 {1,2} rejected



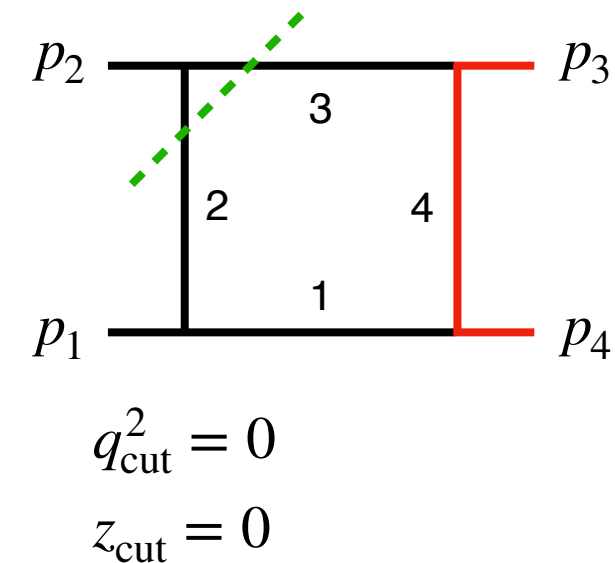
3 {3,4} rejected



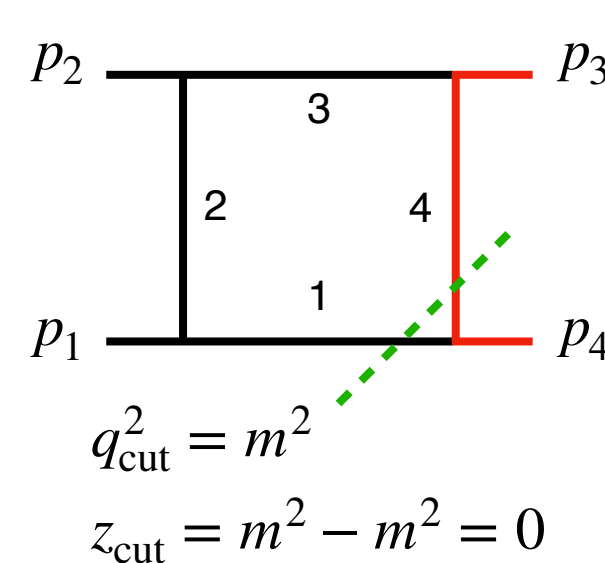
5 {1,3} selected



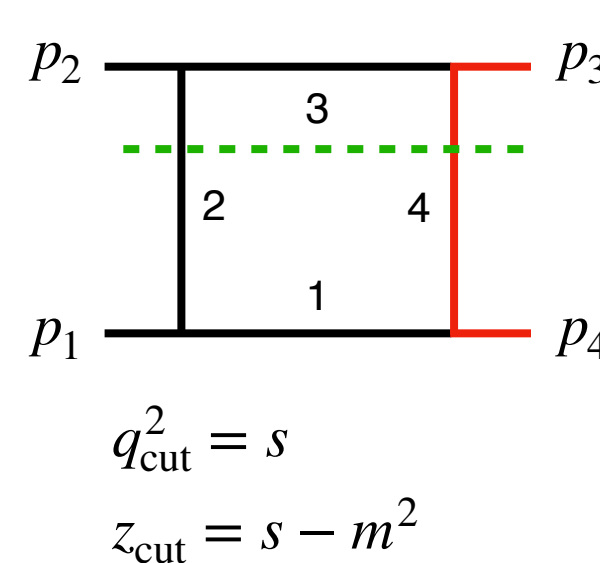
2 {2,3} rejected



4 {1,4} rejected

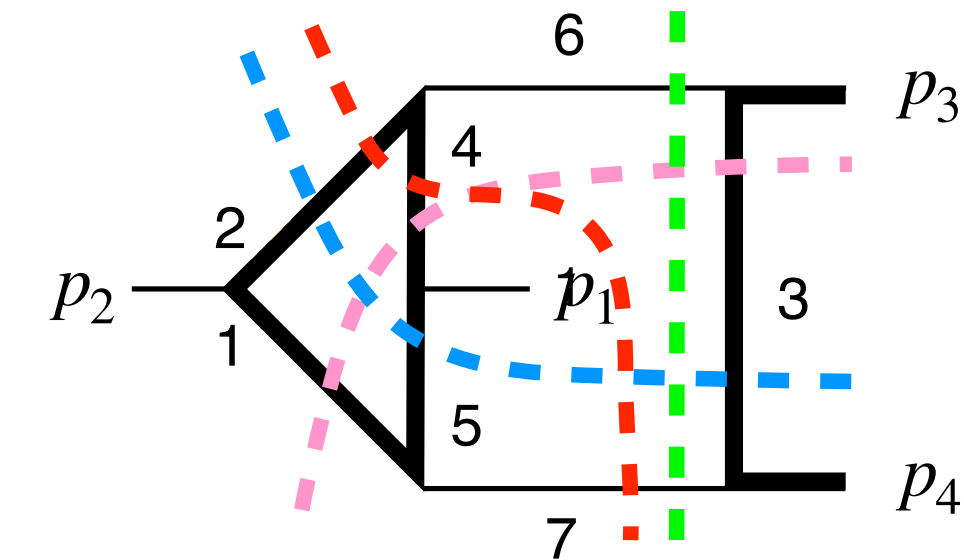


6 {2,4} selected



EXAMPLE 2

2-LOOP NON-PLANAR BOX WITH A MASS



Same kinematics

tot-branch: 4
massless-branch: 1
branches:

- - s
- - $s - 4m^2$
- - $t - 9m^2$
- - $-s - t - 7m^2$

Cutkosky cuts: 17

cut 0: {1, 2}, $q = p_2$, $q^2 = 0$

cut 1: {3, 6}, $q = -p_3$, $q^2 = m^2$

cut 2: {3, 7}, $q = -p_1 - p_2 - p_3$, $q^2 = m^2$

cut 3: {4, 5}, $q = p_1$, $q^2 = 0$

cut 4: {6, 7}, $q = -p_1 - p_2$, $q^2 = s$

cut 5: {1, 3, 4}, $q = p_2 + p_3$, $q^2 = t$

cut 6: {1, 3, 5}, $q = p_1 + p_2 + p_3$, $q^2 = m^2$

cut 7: {1, 4, 6}, $q = p_2$, $q^2 = 0$

cut 8: {1, 4, 7}, $q = -p_1$, $q^2 = 0$

cut 9: {1, 5, 6}, $q = p_1 + p_2$, $q^2 = s$

cut 10: {1, 5, 7}, $q = 0$, $q^2 = 0$

cut 11: {2, 3, 4}, $q = p_3$, $q^2 = m^2$

cut 12: {2, 3, 5}, $q = p_1 + p_3$, $q^2 = 2m^2 - s - t$

cut 13: {2, 4, 6}, $q = 0$, $q^2 = 0$

cut 14: {2, 4, 7}, $q = -p_1 - p_2$, $q^2 = s$

cut 15: {2, 5, 6}, $q = p_1$, $q^2 = 0$

cut 16: {2, 5, 7}, $q = -p_2$, $q^2 = 0$

Kinematic bridge

When finding the branch cuts, **LINE** adopts a fixed internal notation for the kinematics

config/kinematics.yaml

```
kinematics:
  incoming_momenta: [p1, p2, p3, p4,]
  outgoing_momenta: []
  momentum_conservation: [p4, -p1-p2-p3]
  masses: []
  kinematic_invariants: [s, t, mt2]
  scalarproduct_rules:
    - [[p1, p1], 0],
    - [[p2, p2], 0],
    - [[p3, p3], mt2],
    - [[p1+p2, p1+p2], s],
    - [[p2+p3, p2+p3], t],
    - [[p1+p3, p1+p3], -s-t+2*mt2]
```

User-defined kinematics

Internal notation for the kinematics

$$s_{ij} \equiv 2 p_i \cdot p_j$$

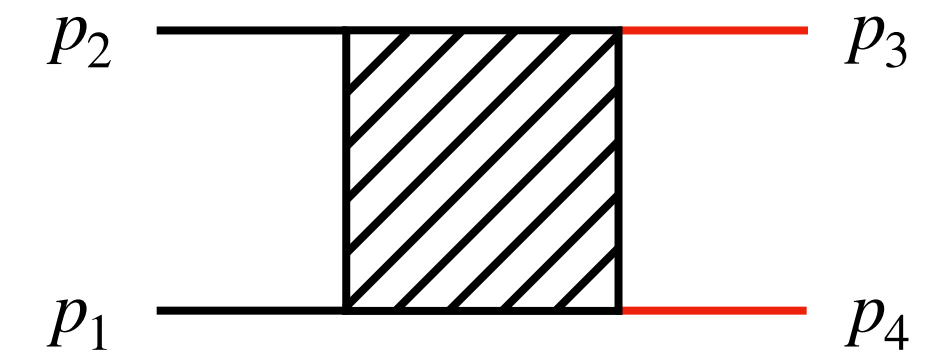
$$t_{ij} \equiv (p_i + p_j)^2 = p_i^2 + p_j^2 + s_{ij}$$



stdout

```
s12 = s
t12 = s
s13 = mt2-s-t
t13 = 2*mt2-s-t
s14 = -mt2+t
t14 = t
s23 = -mt2+t
t23 = t
s24 = mt2-s-t
t24 = 2*mt2-s-t
s34 = -2*mt2+s
t34 = s
```

Kinematic bridge



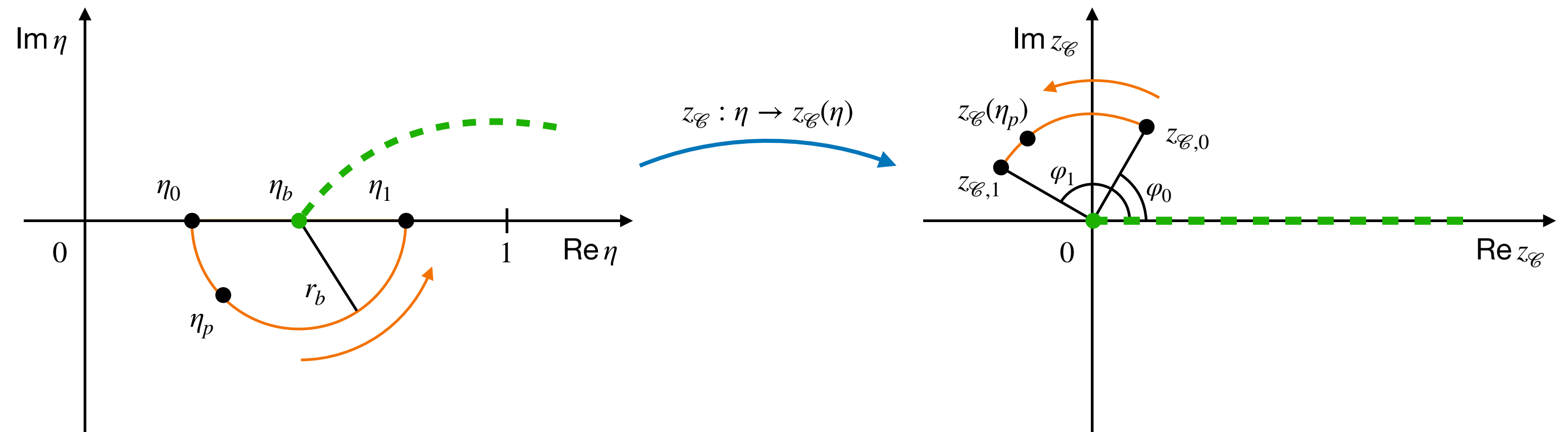
- A set of independent invariants is chosen in the internal notation
- The **scalarproduct_rules** are collected to find a set of independent relations
- These are inverted to express the internal kinematic invariants in terms of the user-defined ones

Detour around singular points

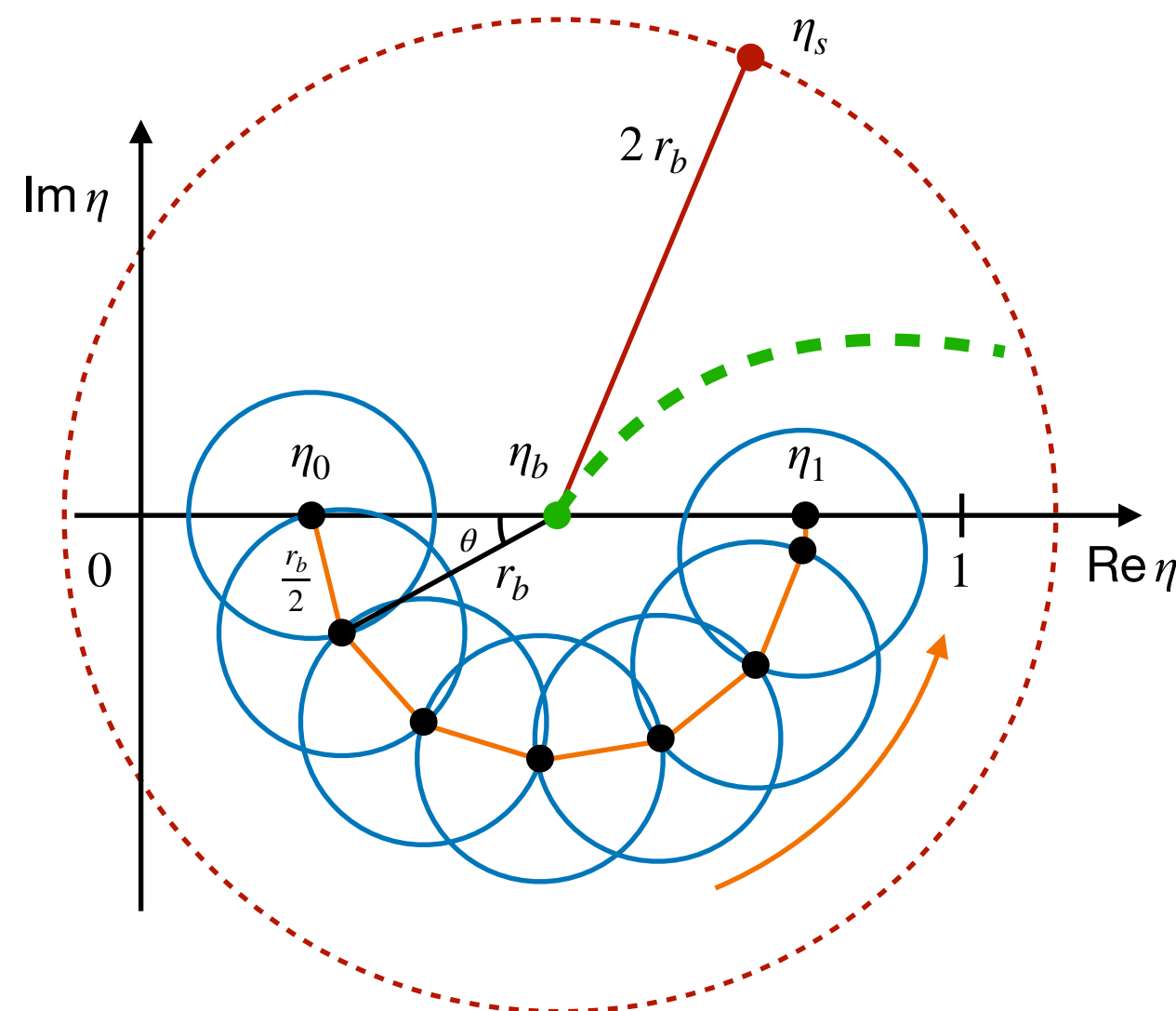
With **LINE**'s prescription for **analytic continuation** comes naturally an **alternative strategy** to get past **singular points**

*Which half of the complex plane allows one to land on the branch from the side selected by the **Feynman's prescription**?*

The answer gives the proper sign for the imaginary parts of logarithms



BUT the solution can be actually propagated by taking a **detour** on that side of the complex plane → **only regular steps** are required!



WHY IT WORKS

- η_s : singularity closest to η_b
- $2 r_b$: radius of convergence around η_b



- η_0 has η_b as closest singularity → step size is $r_b/2$
- we move on the circle of radius r_b
- all the detour points maintain η_b as closest singularity

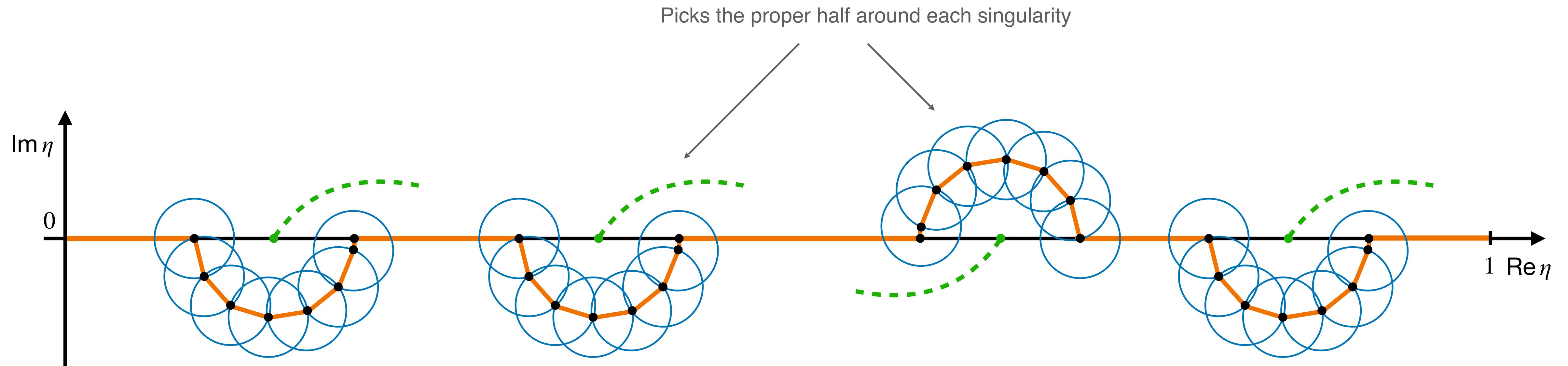
\$ line --singularity-detour



$\theta = \arccos(7/8) \simeq \pi/6.22$
 6 intermediate points
 Detour of 7 regular regular steps

Detour around singular points

```
$ line --singularity-detour
```

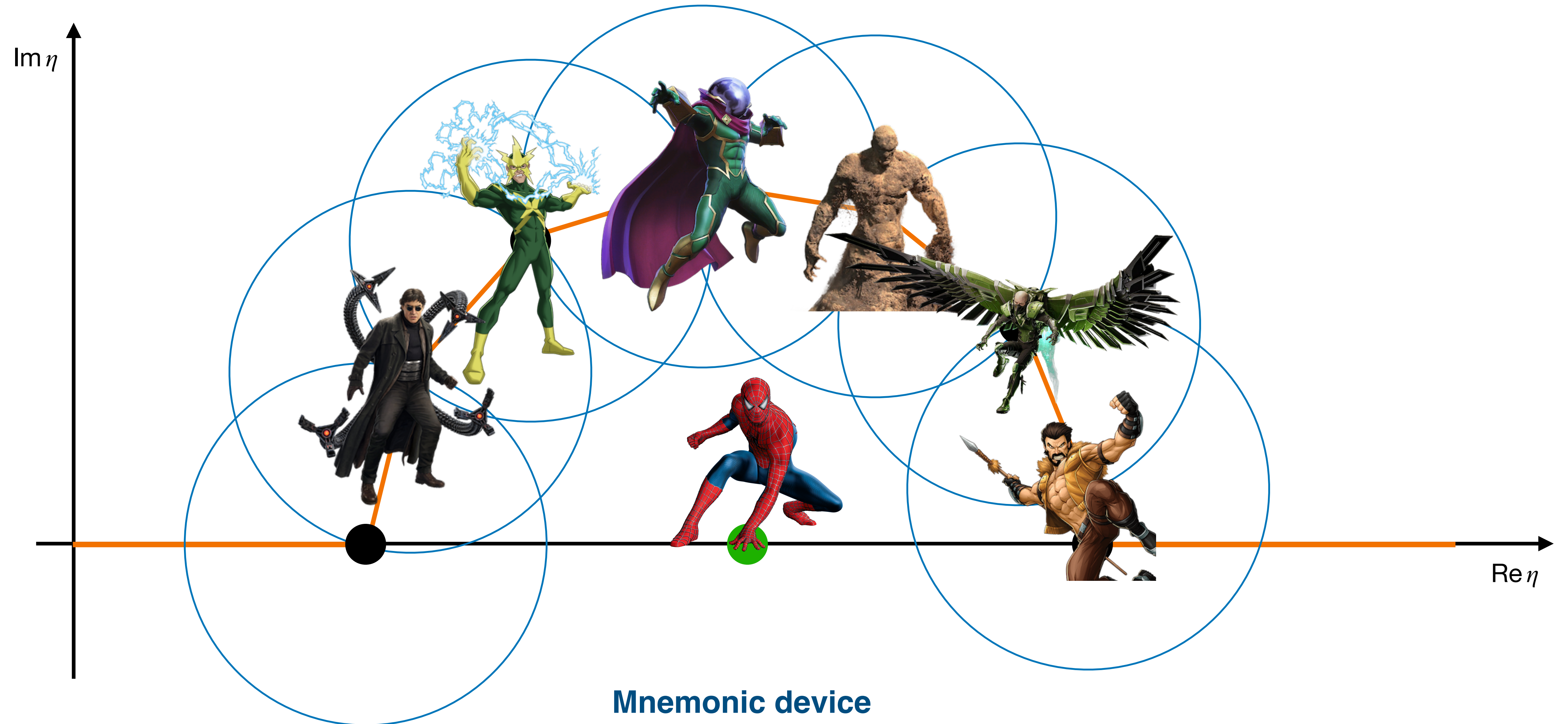


Extends the toolkit to deal with singularities

- **Double-check** results (singular expansion vs detour)
- **Fallback** mechanism if singular expansion is problematic

Makes **LINE** more robust

Detour around singular points



Spider-Man and related characters © Marvel.
Images used for illustrative, non-commercial purposes.

Enhanced solution caching

Besides saving the solution at the target point, **LINE** can now store/compare the MIs at any intermediate path point

```
$ line --write-path-pts 1
```

```
points/  
├── hash-pt1/  
│   ├── card.yaml  
│   └── sol/  
├── hash-pt2/  
├── hash-pt3/  
└── hash-pt3/
```

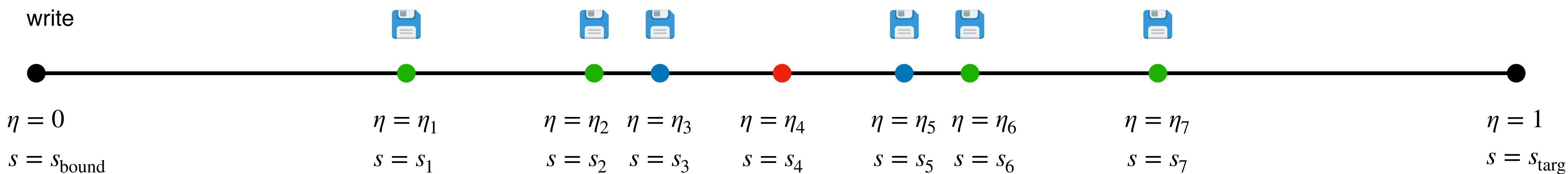
card.yaml can be used as input card to reach any path point from its previous neighbour

useful **debug tool**

- OPTION `--write-path-pts`
- `-1` (default) do nothing
 - `0` compare
 - `1` write

```
$ line --write-path-pts 0  
$ line -i path/to/card.yaml -w 0
```

- 1. Locate the problem
- 2. Isolate and solve it



```
hash-pt1/card.yaml  
  
point:  
  s: s_1  
starting-point:  
  s: s_bound
```

```
hash-pt2/card.yaml  
  
point:  
  s: s_2  
starting-point:  
  s: s_1
```

```
hash-pt5/card.yaml  
  
point:  
  s: s_5  
starting-point:  
  s: s_3
```

Every run enriches the pool of **reusable** starting **points**!

↓
fill the phase space
for **Monte Carlo** integration

Automated nearest-point selection

With a few runs of **LINE**, the user can store the MIs computed at many phase space points



They constitute potential starting points for later runs



For a given target point, **LINE** can automatically select the **nearest staring point** among the available candidates

```
$ line --find-starting-pt
```

Makes it easier to compute more points
(no need for manual selection)

CACHE PTS	ORD / PREC	DISTANCE	STATUS
pt1	6 / 30	4	eligible
pt2	6 / 30	1	selected
pt3	4 / 20	0.5	precision differs
pt5	6 / 30	0	target itself

Points are filtered
to select only
valid candidates

Implemented metric:

$$d(P_i, P_j) = \sqrt{(s_i - s_j)^2 + (t_i - t_j)^2 + \dots}$$

Euclidean distance

Extendable with more sophisticated metrics:

- count Cutkosky thresholds that are to be crossed
- count weighted number of regular and singular steps
- ...

Enhanced control over convergence and precision

CONVERGENCE

LINE provides tools to check and ensure the **convergence** of the series solution on all the working-precision digits

```
$ line --check-conv 1
```

OPTION **--check-conv**

-1 (default) check and print a warning

0 do not check

1 check and dynamically enlarge the series

- The inclusion of extra-orders is performed **on the fly** and only the last propagation step is repeated
- The convergence is efficiently estimated accessing only the base-2 exponents of the **mpfr** representation

```
$ line --incr-eta-ord 1.2
```

OPTION **--incr-eta-ord** *value* to multiply times *value* the number of terms in the series expansions

PRECISION

The working precision might be not enough to reach the final point at target precision. **LINE** offers mechanism to detect and cure these cases

```
$ line --check-resid 1 --check-prec 1
```

Check the residual of the
DE after each step

Check the precision of the epsilon
series at the target point

OPTION **--check-resid**

-1 (default) check and print a warning

0 do not check

1 check and abort the run

OPTION **--check-prec**

0 do not check

1 (default) check

The precision of the epsilon interpolation is tested by interpolating with one fewer sample and check the number of stable digits

```
$ line --incr-wp 1.2
```

OPTION **--incr-wp** *value* to multiply times *value* the number of working-precision digits

Conclusions and outlook

- **LINE**: an **open-source** tool to compute **Feynman integrals** (and more) solving **differential equations** via series expansion
- Generation of **boundary** values and their **propagation** across the phase space **in a single tool**
- Flexible input and configuration (**YAML** format, **options** from files and command line, kinematics in **Kira/Reduze** conventions)
- Automated generation of **branch cuts** to cross singularities with proper **analytic continuation**
- **Singularity detour** to increase robustness around singular points
- Enhanced solution **caching** with **nearest-point selection**
- **Self-contained** procedures to check and adjust **precision**

Conclusions and outlook

- **LINE**: an **open-source** tool to compute **Feynman integrals** (and more) solving **differential equations** via series expansion
- Generation of **boundary** values and their **propagation** across the phase space **in a single tool**
- Flexible input and configuration (**YAML** format, **options** from files and command line, kinematics in **Kira/Reduze** conventions)
- Automated generation of **branch cuts** to cross singularities with proper **analytic continuation**
- **Singularity detour** to increase robustness around singular points
- Enhanced solution **caching** with **nearest-point selection**
- **Self-contained** procedures to check and adjust **precision**

WHAT'S NEXT?

- High-level coding structures for inexpensive operations
- Boundary generation beyond two loops → AMFlow recursive strategy
- Performance optimisation
- ... and more!

**TO BE
CONTINUED...** →

Conclusions and outlook

- **LINE**: an **open-source** tool to compute **Feynman integrals** (and more) solving **differential equations** via series expansion
- Generation of **boundary** values and their **propagation** across the phase space **in a single tool**
- Flexible input and configuration (**YAML** format, **options** from files and command line, kinematics in **Kira/Reduze** conventions)
- Automated generation of **branch cuts** to cross singularities with proper **analytic continuation**
- **Singularity detour** to increase robustness around singular points
- Enhanced solution **caching** with **nearest-point selection**
- **Self-contained** procedures to check and adjust **precision**

WHAT'S NEXT?

- High-level coding structures for inexpensive operations
- Boundary generation beyond two loops → AMFlow recursive strategy
- Performance optimisation
- ... and more!

**TO BE
CONTINUED...**

**...STAY
TUNED!**

Backup

Block Strategy

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

$$A = \left(\begin{array}{c|cc|c} A_{11} & 0 & 0 & 0 \\ \hline A_{21} & A_{22} & & 0 \\ \hline A_{31} & A_{32} & & A_{33} \end{array} \right)$$

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

After solving $b-1$ blocks:

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

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

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

known from previous blocks

ε - dependence

DEs depend on the space-time dimension $d = 4 - 2\varepsilon$ (**dimensional regularization**)

- solve DEs for different **numerical values of epsilon**
 - **interpolate** epsilon orders (strategy implemented in the AMFlow package)
- } trade two-variate problem for multiple **independent univariate** problems of **equal complexity** (**parallelization**)

$$\vec{I} = \frac{1}{\epsilon^p} \sum_{k=0}^{\infty} \vec{I}_k \epsilon^k \quad \text{Laurent expansion}$$

$$\varepsilon = \frac{101}{146700}$$

```
MI0: 2.90441164026269689878103960...e3
MI1: 4.35540146043925901671524332...e3
MI2: 7.25644994136370315072624217...e3
MI3: 1.45102408899755166053045532...e3
MI4: 1.45063139101440792324071813...e3
MI5: 1.45053840930241730004028775...e3
MI6: -1.5499533725585654771242419...e-1
```

$$\varepsilon = \frac{17}{24450}$$

```
MI0: 2.87593175196824945061702911...e3
MI1: 4.31268163190383444307933535...e3
MI2: 7.18525024330887846376219687...e3
MI3: 1.43678414841565105694008735...e3
MI4: 1.4363914550165960790255637...e3
MI5: 1.4362984743558711152632678...e3
MI6: -1.5499351276863830613546647...e-1
```

⋮

interpolation

$$A(\varepsilon, \eta) \rightarrow A(\varepsilon_1, \eta), A(\varepsilon_2, \eta), \dots$$

MI5 bubble
eps^-2: 0
eps^-1: 9.999999999999999999999999e-1
eps^0: -1.938704283006374112859e0
eps^1: 2.712629538677921993042e0
eps^2: -3.230718213168225851725e0
eps^3: 3.549244487845649233171e0
eps^4: -3.732208990036951965128e0
eps^5: 3.832940123529834150758e0

MI6 triangle
eps^-2: 0
eps^-1: 0
eps^0: -1.55179783617978100156e-1
eps^1: 2.68153011739011483628e-1
eps^2: -3.62299367768517539729e-1
eps^3: 4.21339386265382316643e-1
eps^4: -4.57066908219392659241e-1
eps^5: 4.77147594900988835882e-1

ε - dependence

Relative error on the coefficients

$$E_k = \frac{|\hat{I}_k - I_k|}{|I_k|} \sim \left(\frac{r}{\mathcal{R}}\right)^{N_\varepsilon - k} \quad \longleftarrow \text{the error increases order by order}$$

$$r/\mathcal{R} \sim \varepsilon$$

the leading order must be known
with higher (working) precision **wp**

ask for the higher order to have
the target precision

$$E_{\text{ord}} \lesssim \left(\frac{r}{\mathcal{R}}\right)^{N_\varepsilon - \text{ord}} \sim E_{\text{prec}}$$

Choose N_ε and wp to minimize the computational time $\Delta t \sim N_\varepsilon (\text{wp})^\alpha$

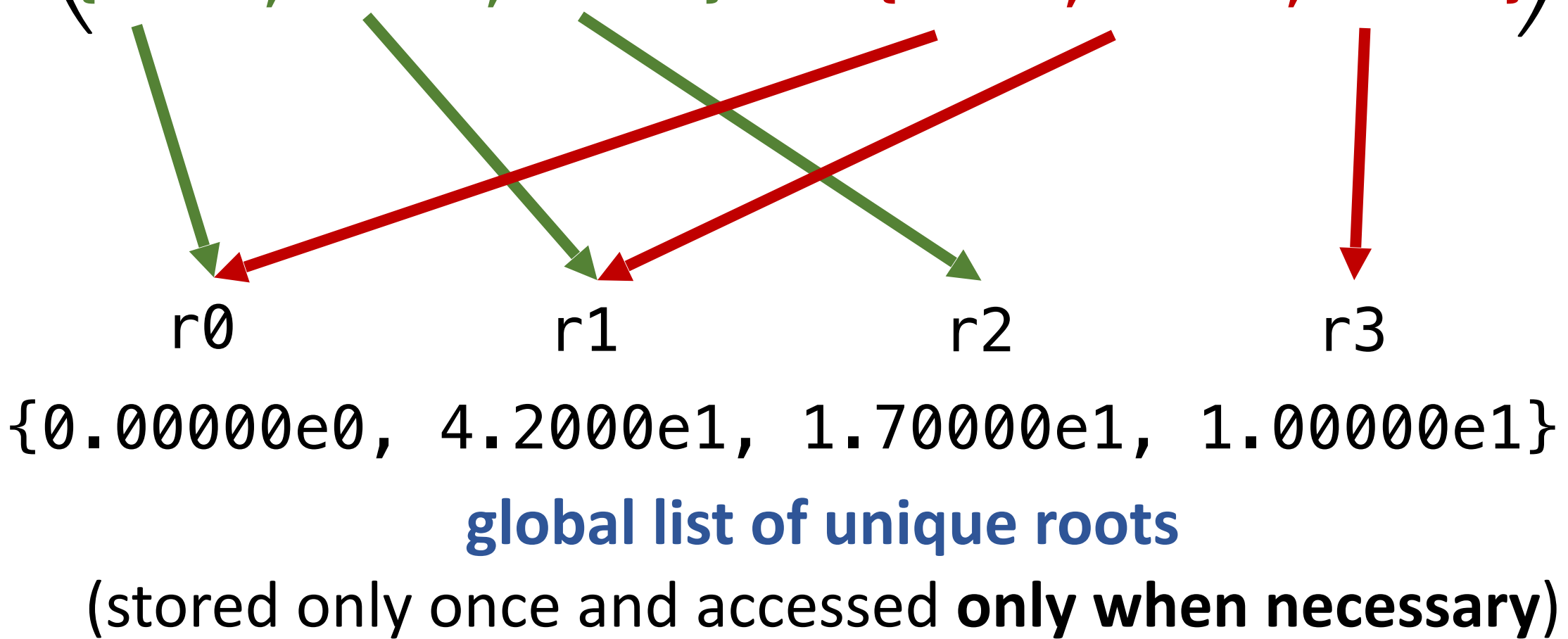
$$N_\varepsilon = (1 + \alpha) \text{ord}, \quad \text{wp} = \left(\frac{1 + \alpha}{\alpha}\right) \text{prec}$$

Rational Functions

$$\left(\begin{array}{ccc} \frac{N_{11}(\eta)}{\eta(\eta-42)^3} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{N_{n1}(\eta)}{\eta^2(\eta-42)^2(\eta-17)^4} & \cdots & \frac{N_{nn}(\eta)}{\eta(\eta-42)^3(\eta-10)^2} \end{array} \right) \left(\begin{array}{ccc} \{r0: 1, r1: 3\} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \{r0: 2, r1: 2, r2: 4\} \cdots \{r0: 1, r1: 3, r3: 2\} \end{array} \right)$$

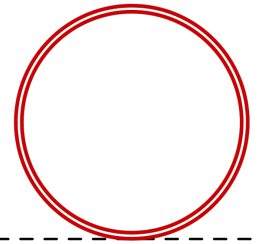
For each denominator:

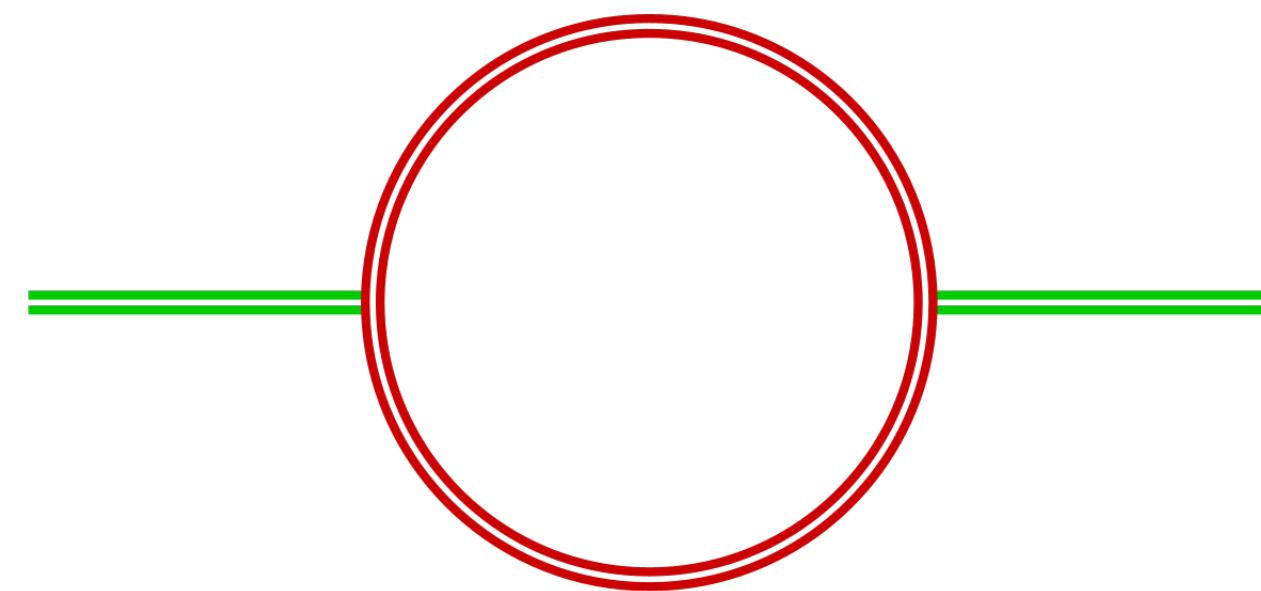
- find roots **numerically** with **arbitrary precision**
- updated a list of **unique roots**
- assign a **label** to each root
- represent each denominator with **integer labels** and **multiplicities**



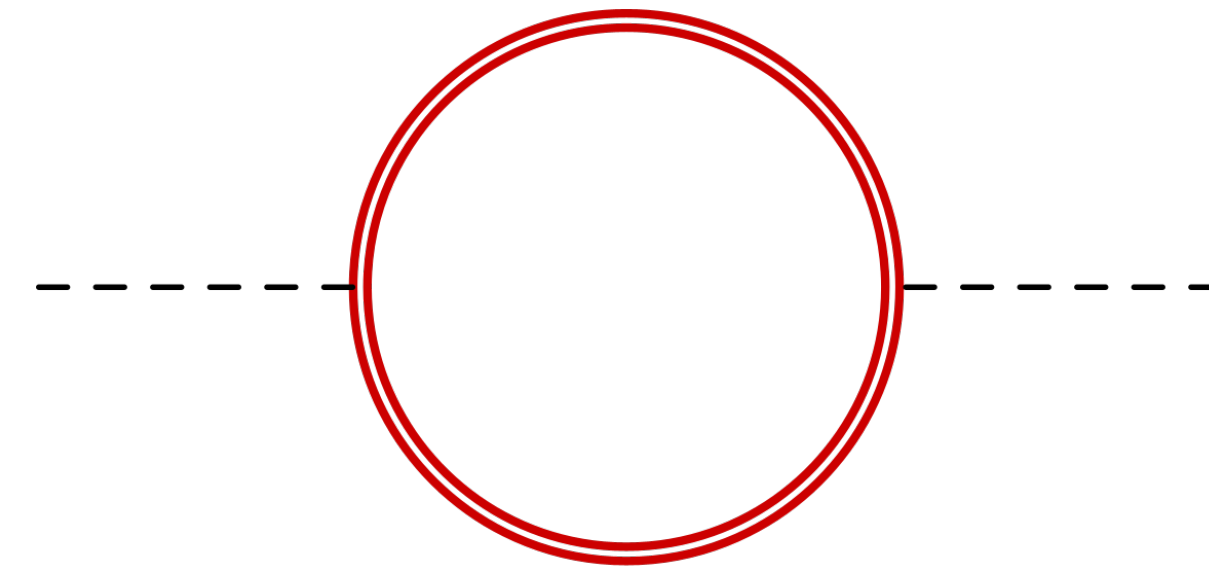
fast computation of polynomial LCM, shifts, simplifications
(only deal with small integers)

Boundary Conditions via Expansion by Regions

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



$s \rightarrow 0$



know **regular**
contribution

DE for the bubble

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

must be
regular

regular

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



$$B(0) = -\frac{f_{B,A}(0)}{f_{B,B}(0)} A$$

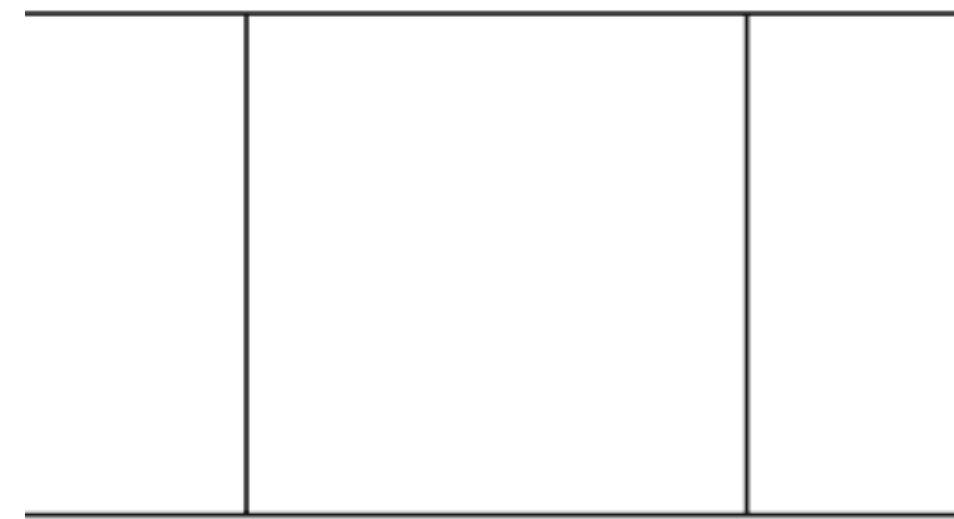
$$m^{-2\varepsilon}(\varepsilon - 1)\Gamma(\varepsilon - 1) = -\frac{(\varepsilon - 1)}{m^2} A$$

can be obtained from the DE

Boundary Conditions via Expansion by Regions

1-loop massless box in the limit $u \rightarrow 0$: only the **bubble** is needed

$$u = 0 \iff s = -t$$



$$u \rightarrow 0$$



regular

DE for the box

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

regular

s - and t -channel bubbles
(subsectors)

$$u \propto \eta$$

must be
regular

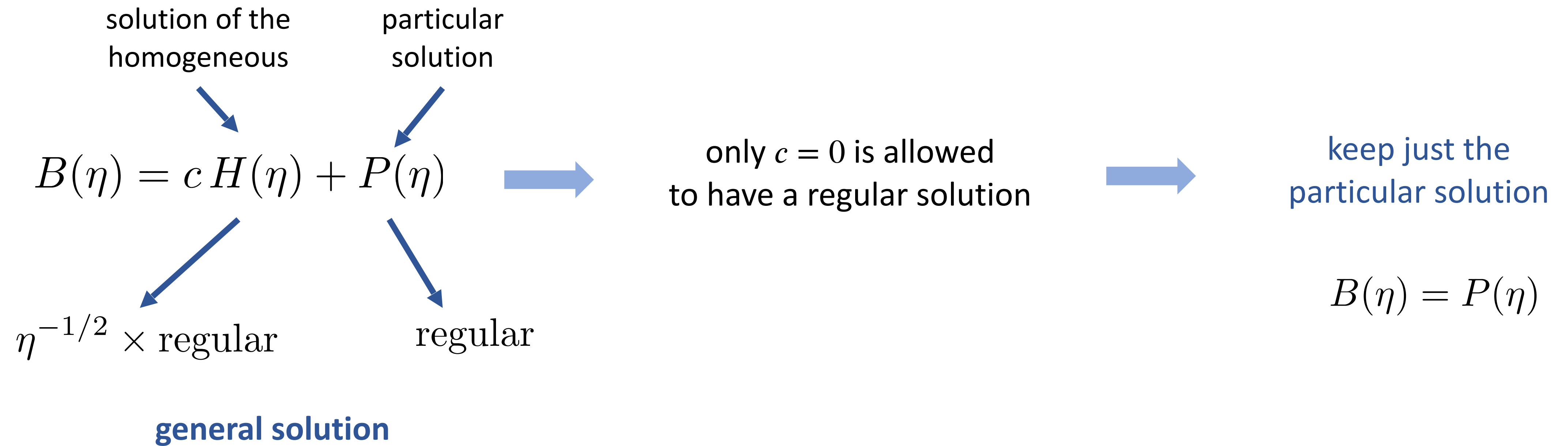
$$0 = \frac{f_{B,B}(0)}{\eta} B(0) + \frac{f_{B,A}(0)}{\eta} A$$



$$B(0) = -\frac{f_{B,A}(0)}{f_{B,B}(0)} A$$

Boundary Conditions via Expansion by Regions

The implementation is even simpler



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

Solving around a Regular Singular Point

$$\tilde{\mathbb{H}}(\eta) = \mathbb{P}(\eta) \mathbb{Z}(\eta) \quad \text{fundamental matrix solution (columns are independent solutions)}$$

regular, with
 $\mathbb{P}(0) = \mathbb{I}$

solution of

$$\frac{d}{d\eta} \mathbb{Z}(\eta) = \frac{\tilde{\mathbb{A}}_0}{\eta} \mathbb{Z}(\eta)$$

(captures the singular behaviour)

$$\mathbb{Z}(\eta) = \eta^{\tilde{\mathbb{A}}_0} = e^{\tilde{\mathbb{A}}_0 \log(\eta)} = \sum_{\lambda \in \Lambda} \eta^\lambda \sum_{k=0}^{K_\lambda} \frac{\log^k(\eta)}{k!} \mathbb{Z}_{\lambda,k}$$

$$\tilde{\mathbb{I}}(\eta) = \sum_{\lambda \in \Lambda} \eta^\lambda \sum_{k=0}^{K_\lambda} \frac{\log^k(\eta)}{k!} \sum_{n=0}^{\infty} \tilde{\mathbb{I}}_{\lambda,k,n} \eta^n$$

Jordan Normal Form

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

Jordan normal form:

Jordan blocks

$$A'_0 = \begin{pmatrix} \boxed{\lambda_1} & \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{\lambda_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{\lambda_1} & \boxed{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{\lambda_1} & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{\lambda_1} & \boxed{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{\lambda_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \boxed{\lambda_2} & \boxed{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{\lambda_2} & \boxed{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{\lambda_2} \end{pmatrix}$$

heads of the chains

Jordan chain:

$$A'_0 \vec{v}_1 = \lambda \vec{v}_1 \quad \text{head of the chain (eigenvector)}$$

$$A'_0 \vec{v}_i = \lambda \vec{v}_i + \vec{v}_{i-1} \quad i = 2, \dots, L$$

generalized eigenvectors

defective eigenvalues: g.m. $\neq$ a.m.

$$\lambda_1: \text{g.m.} = 2, \text{ a.m.} = 6$$

$$\lambda_2: \text{g.m.} = 1, \text{ a.m.} = 3$$

ill-defined numerical problem:

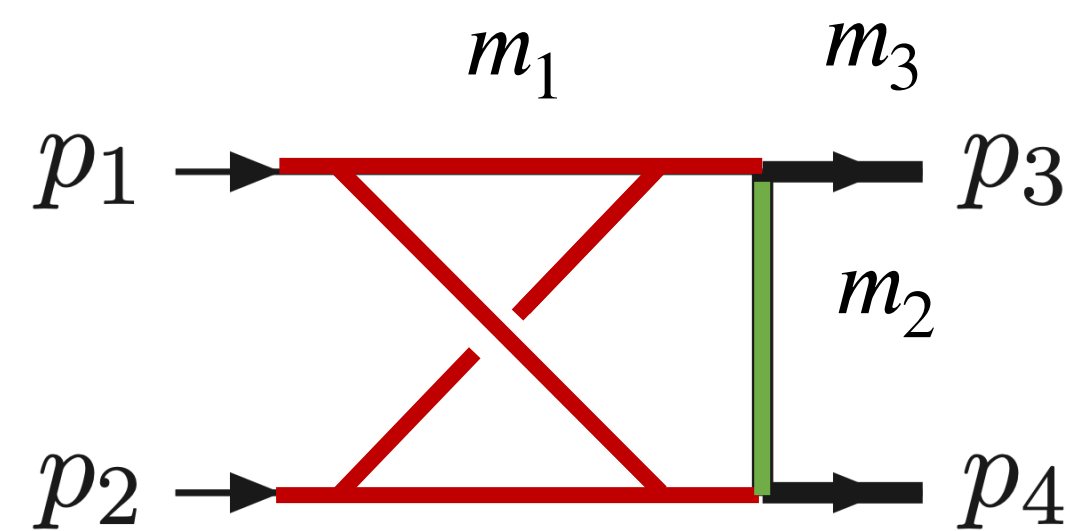
the structure of the output is affected by the **round-off error!**

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

it depends on the precision!

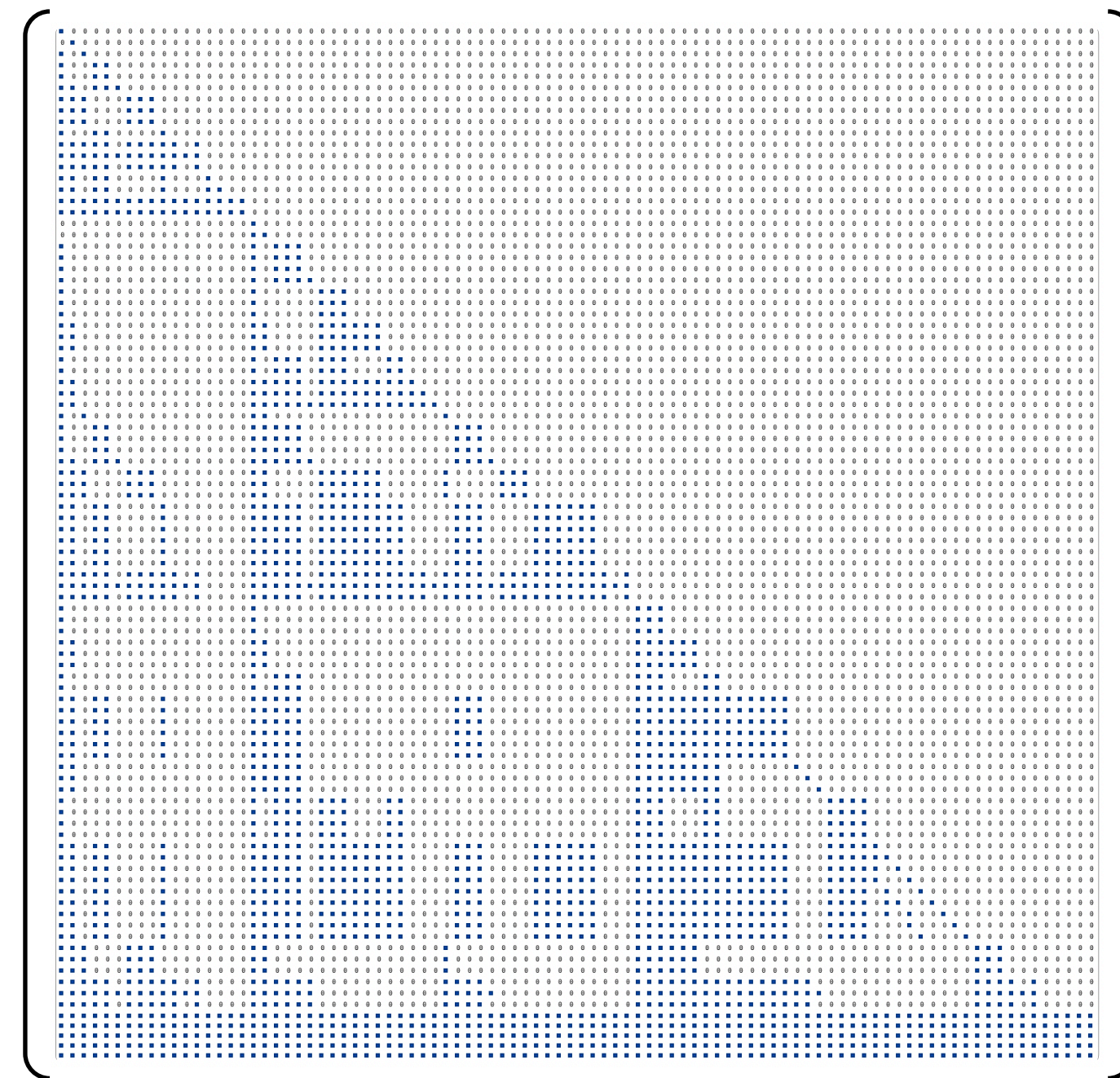
Change to Fuchsian Basis

- 2-loop non-planar double-box with a mass $\rightarrow$ 92 MIs



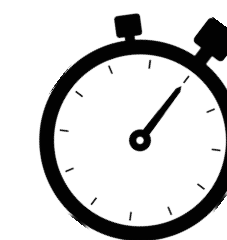
(two internal varying masses,
massless initial state,
massive final state)

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** ✓

Cutkosky Cutting Rules

$$\text{Disc}F(\mathbf{s}) = \sum_{\mathcal{C}} \int_{k_1, \dots, k_L} \prod_{e \in \mathcal{C}} (-2\pi i) \delta(q_e^2 - m_e^2) \theta_H^*(q_e^0) \prod_{e' \in \mathcal{G} \setminus \mathcal{C}} \frac{1}{q_{e'}^2 - m_{e'}^2 + i0^+}$$

Cutkosky cuts

ways to cut the diagram into
two disconnected componets

set on-shell the cut propagators

For the discontinuity to pick up non-vanishing contributions:

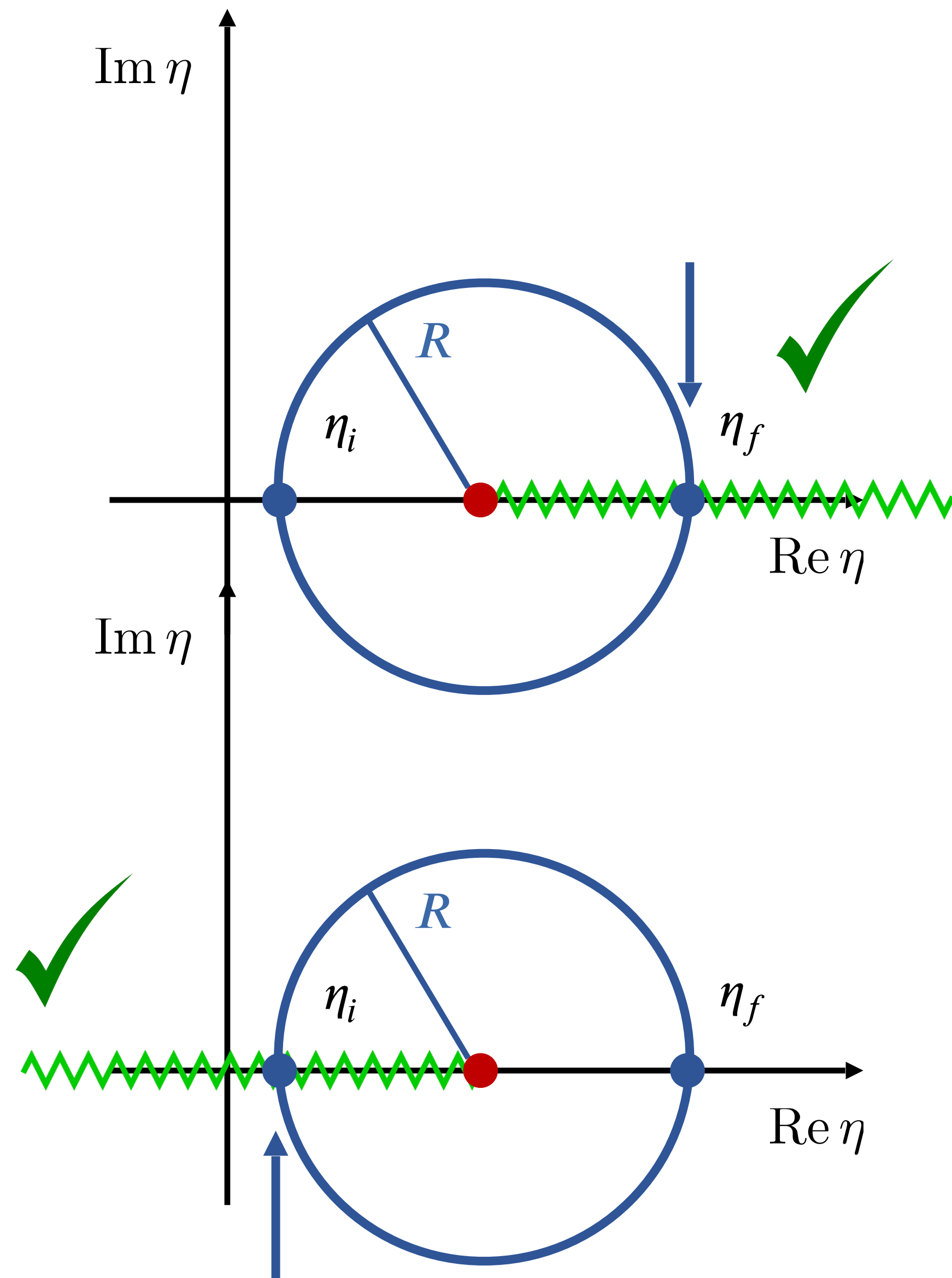
$$z \rightarrow z + i\varepsilon, \quad \varepsilon \rightarrow 0^+$$

squared momentum
flowing through the cut

$$z_{\mathcal{C}} \equiv t_{\mathcal{C}} - \left(\sum_{e \in \mathcal{C}} m_e \right)^2 \geq 0$$

identifies the branch cut

Analytic Continuation – Real Fixed Masses



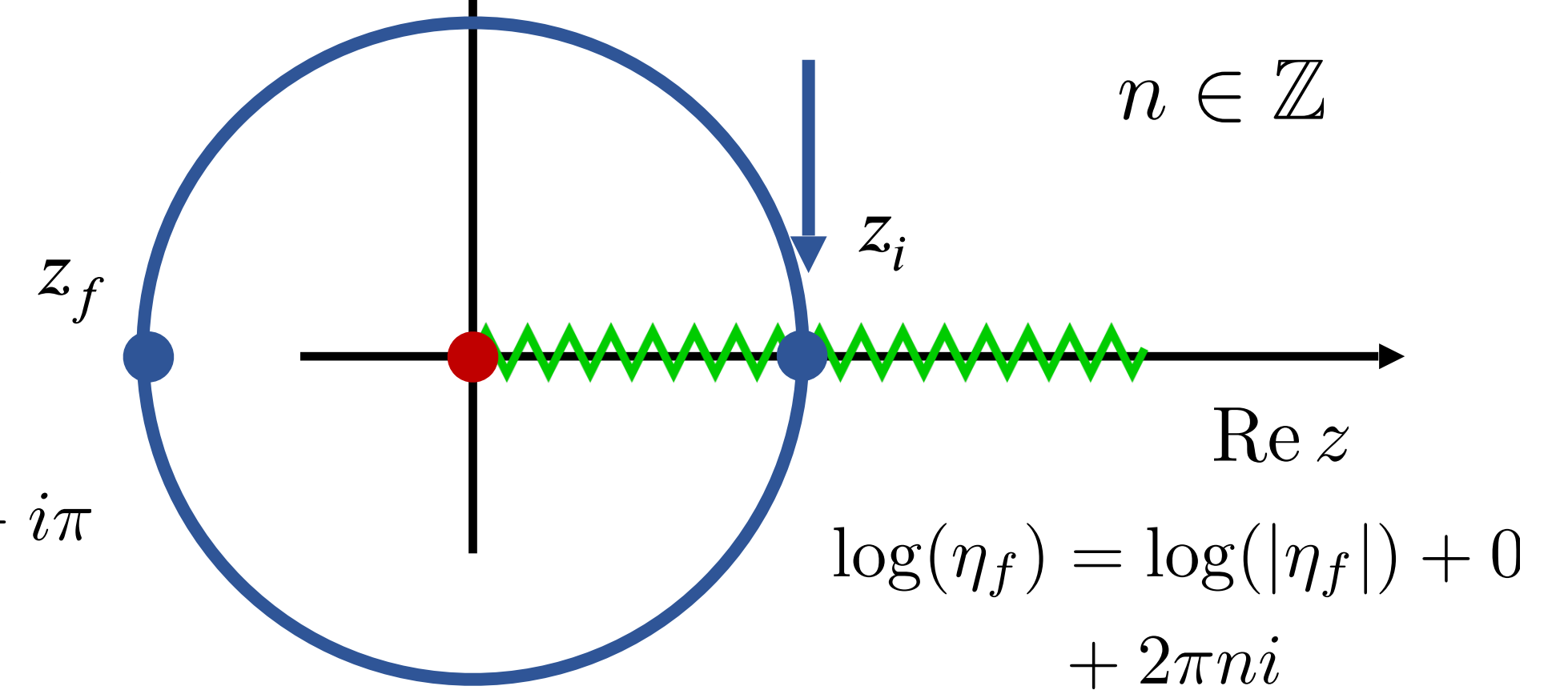
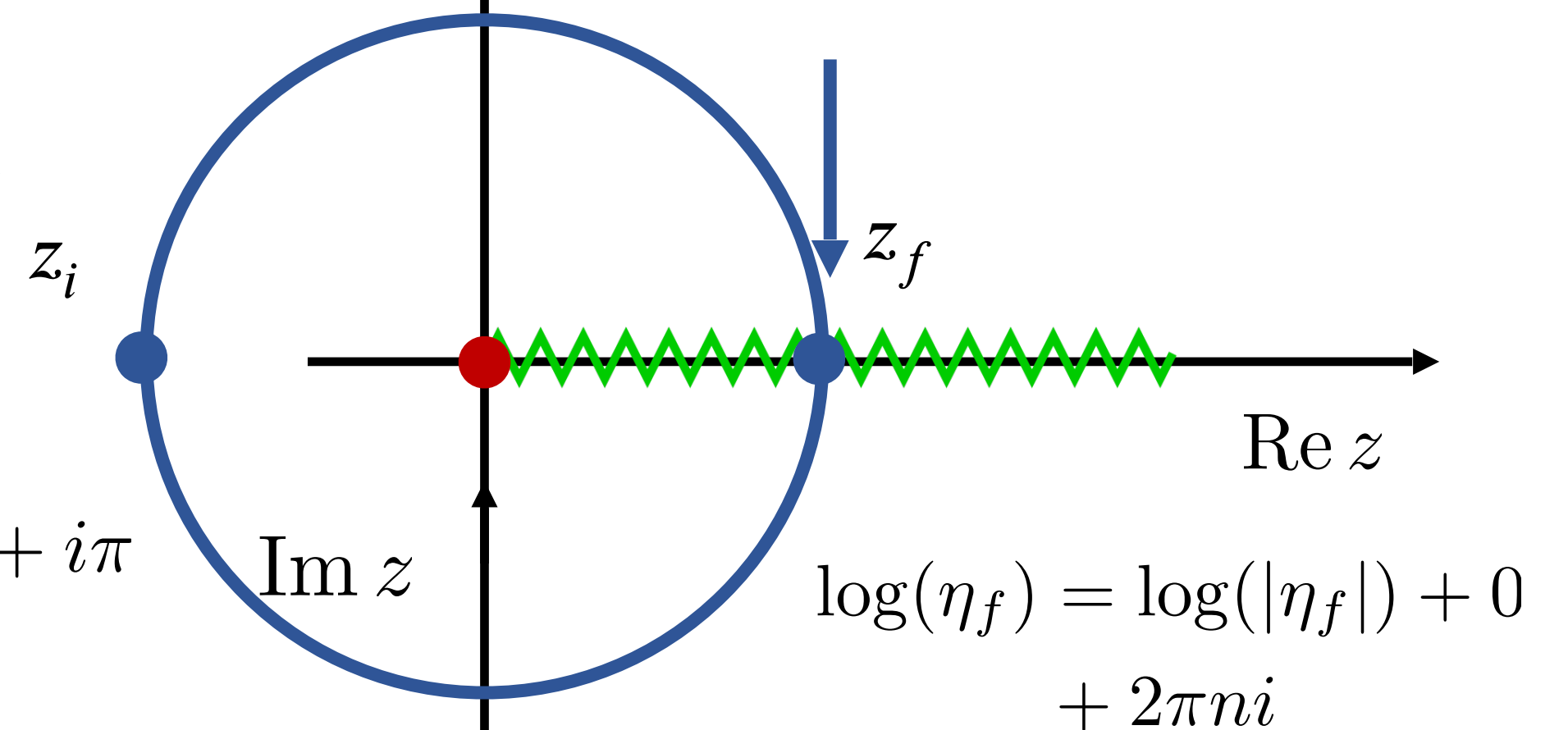
$$z : \eta \rightarrow z(\eta)$$

$$\log(\eta_i) = \log(|\eta_i|) + i\pi + 2\pi ni$$

$$z : \eta \rightarrow z(\eta)$$

$$\log(\eta_i) = \log(|\eta_i|) - i\pi + 2\pi ni$$

$z : \eta \rightarrow z(\eta)$
preserves rotations



Analytic Continuation – General Case

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

$$z(\eta) = c_1 s_1(\eta) + c_2 s_2(\eta) + \cdots \\ - \left[c'_1 \sqrt{m_{1,i}^2 + \eta(m_{1,f}^2 - m_{1,i}^2)} + c'_2 \sqrt{m_{2,i}^2 + \eta(m_{2,f}^2 - m_{2,i}^2)} + \cdots \right]^2$$

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**

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

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

Analytic Continuation – General Case

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

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



Imagine to rotate η counter-clockwise from η_i to η_f :

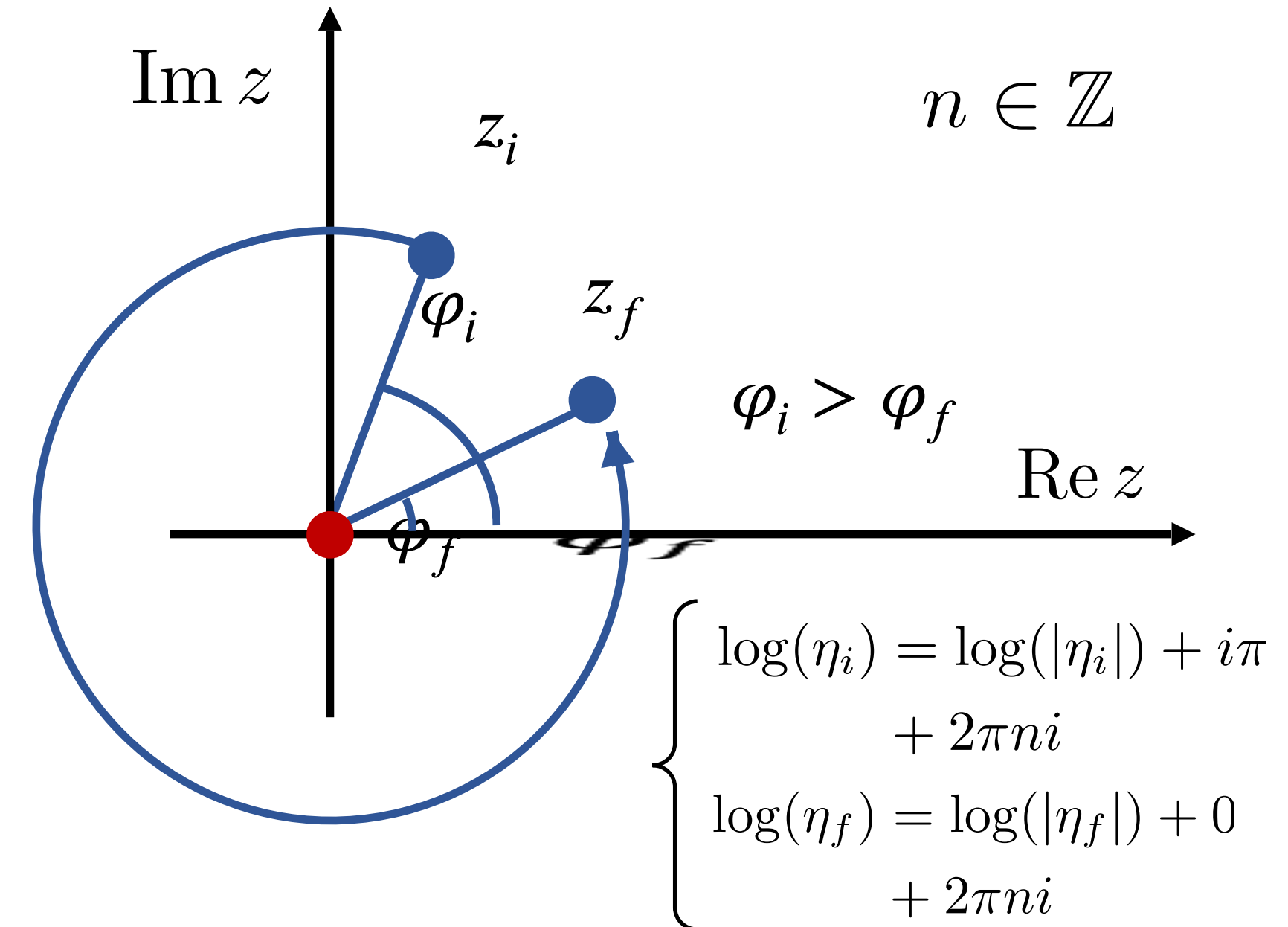
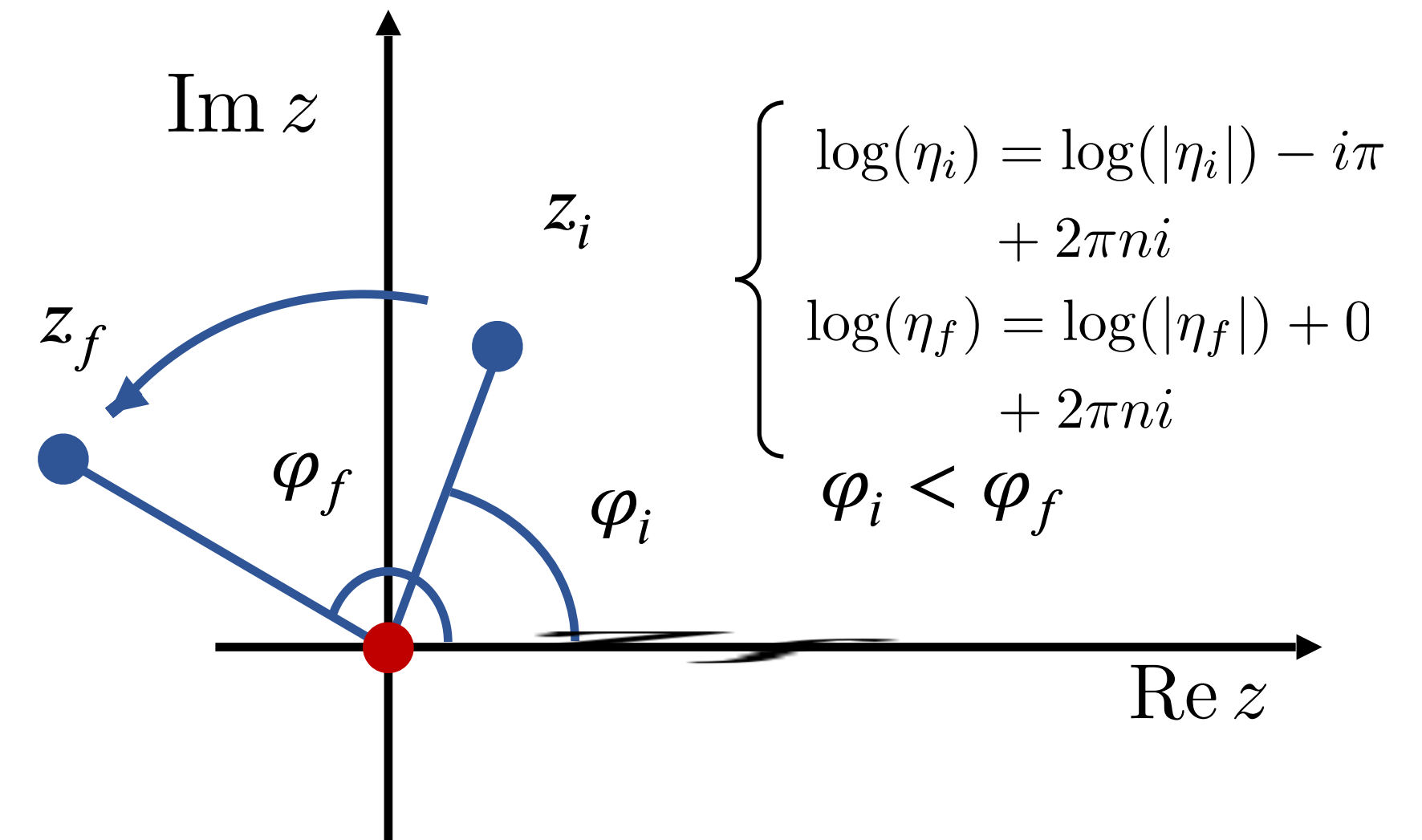
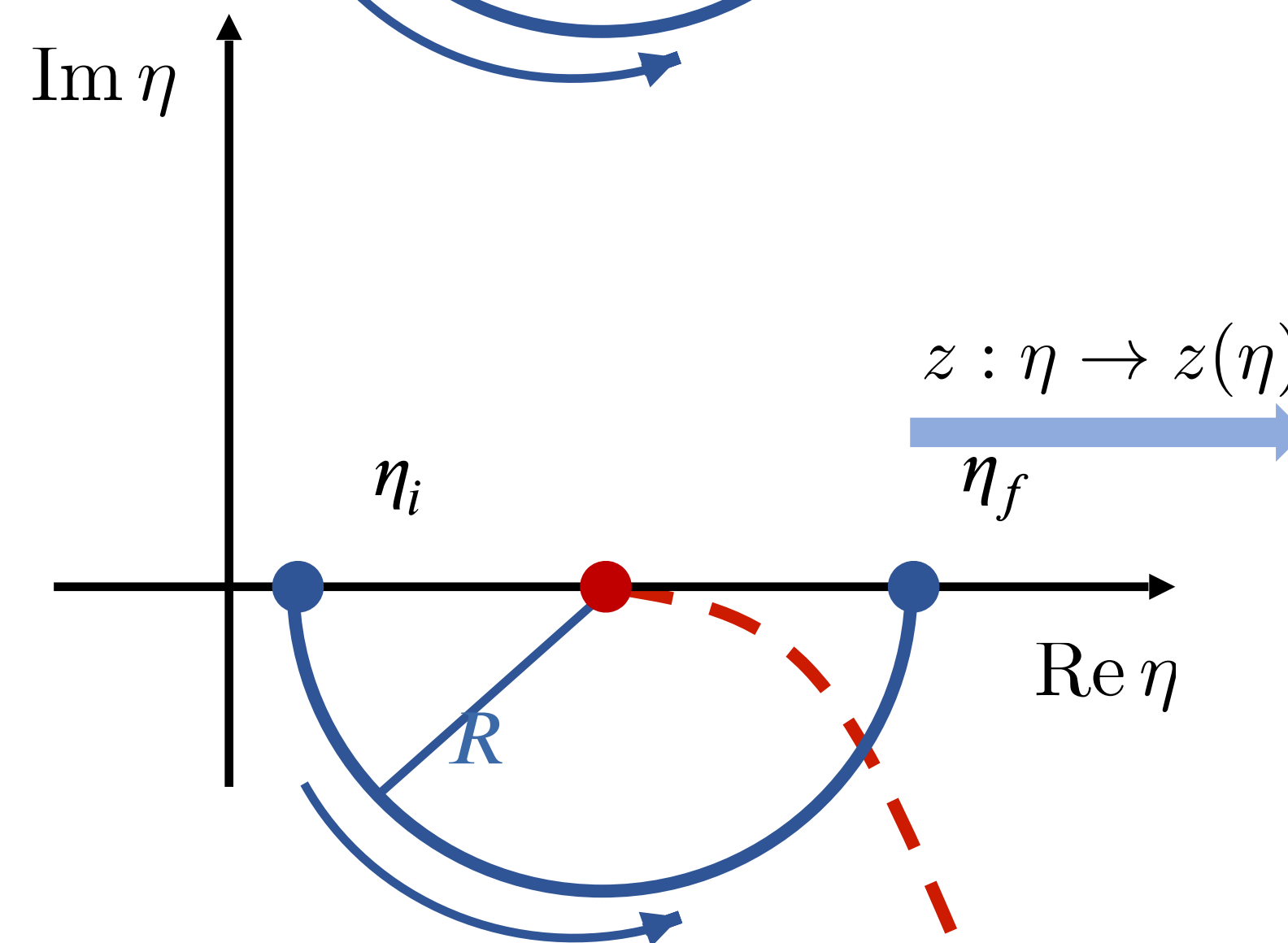
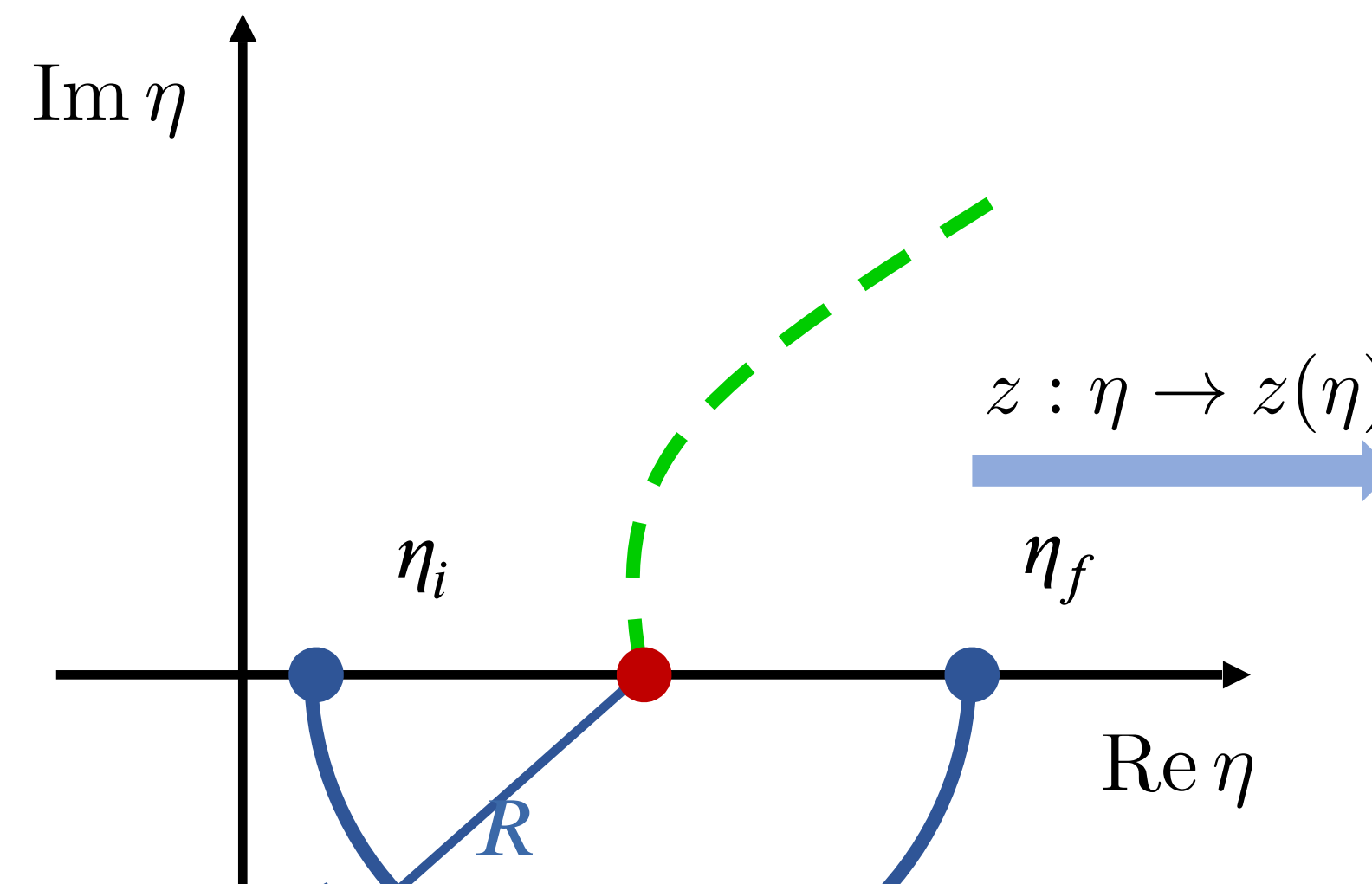
the branch cut is crossed



z crosses the positive real axis



$$\varphi_i > \varphi_f$$



Higher-Loop (> 2)

- **phase-space propagation (with external BCs):** already there in principle
- **automated BCs with EBR ($\# \text{loops} \geq 2$):** generalization under investigation
- **LINE implementation of AMFlow method:**

analytic formulas for vacuum integrals only up to two loops

implementation of the AMFlow iterative strategy planned