

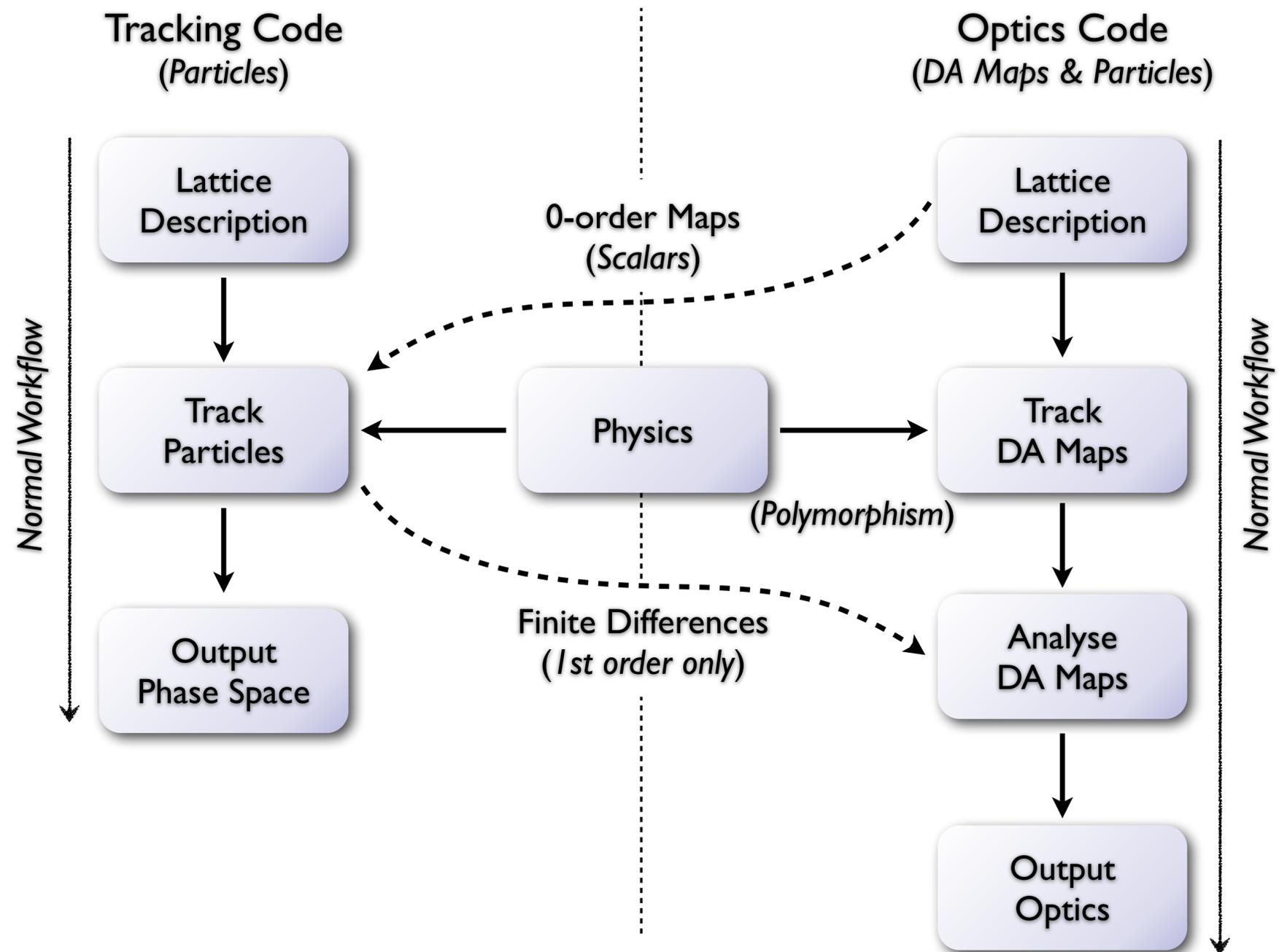
# High-Order Differential Algebra with GTPSA and Its Application in MAD-NG.

28th EuroAD Workshop — CERN  
Laurent Deniau — BE/ABP

9th December 2025

# Tracking Code vs. Optics Code

- Tracking codes **track** (many) *particles*
  - Simple architecture (*scalars*).
    - Use finite differences approximations.
  - Validate existing / operational machines, explore new models.
- Optics codes **track differential maps**
  - Complex architecture (*functions*).
  - Machine design and optimisation.
  - Compact and flexible description.
  - Classes, circuits, optics logic (knobs).
    - e.g. inheritance, deferred expressions.



# Motivation: Why Differential Algebra?

- Phase space:  $z = (q, p; s)$  where  $s$  is the independent variable.
  - $q_i$  and  $p_i$  are  $n$  canonical pairs of variables, e.g. positions and momenta.
- System's Hamiltonian:  $H(q, p; s)$ .  $H(x, p_x, y, p_y, t, p_t) = -(1 + h_s x) \sqrt{(1 + \frac{2p_t}{\beta_0} + p_t)^2 - (p_x - a_x)^2 - (p_y - a_y)^2} + a_s$
- Phase space evolution:  $\frac{dz}{ds} = \{z, H\}$  ; Hamilton's equations are  $2n$  first order PDE.
- Symplectic structure:  $\{f, g\} = \sum_i \left( \frac{\partial f}{\partial q_i} \frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i} \frac{\partial g}{\partial q_i} \right)$ .
  - $\{\cdot, \cdot\}$  are the Poisson brackets, and  $f(z; s)$  and  $g(z; s)$  are functions of the phase space.
  - $\{\cdot, \cdot\}$  define the Lie product, and the Hamiltonian flow invariants  $\{\cdot, H\} = 0$  (Liouville's Theorem).
- Phase space propagator:  $z(s) = e^{\{\cdot, H\}} z(s_0) = \mathcal{M}(z; s) z(s_0)$ .
- Lie operators:  $e^{\{\cdot, H\}} z = z + \{z, H\} + \frac{1}{2!} \{\{z, H\}, H\} + \dots = \sum_{k=0}^{\infty} \frac{1}{k!} \{z, H\}^k$ , (idem for log).

# Differential Algebra — Concepts

- **Taylor Series:**  
(*Functions*)

$$T_f(x; a) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \dots$$

$$+ \frac{f^{(k)}(a)}{k!}(x - a)^k + \dots = \sum_{k=0}^{\infty} \frac{f_a^{(k)}}{k!}(x - a)^k.$$
- **Taylor Polynomial:**  
(Truncated *Functions*)

$$T_f^n(x; a) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \dots + \frac{f^{(n)}(a)}{n!}(x - a)^n$$
- **Finite Differences:**  
(*Scalars*)

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \dots + \frac{h^n}{n!}f^{(n)}(x)$$

$$f'(a) \approx \left. \frac{f(x + h) - f(x - h)}{2h} \right|_{x=a}$$

$$f''(a) \approx \left. \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} \right|_{x=a}$$
  - Finite Differences approximations don't scale for **high-orders** or **multivariate** functions or “near” **singularities** (e.g. abs, inv, sqrt, log, resonance, ...).

# Differential Algebra — Concepts

- Multivariate Taylor Polynomial (TPSA):  
(Truncated **Functions**)

TPSA = Truncated Power Series Algebra.

$$T_f^n(X; A) = \sum_{k=0}^n \frac{f_A^{(k)}}{k!} (X; A)^k = \underbrace{\sum_{k=0}^n \frac{1}{k!}}_{\text{orders}} \underbrace{\sum_{|\vec{m}|=k} \binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}} \Big|_A}_{\text{homogeneous polynomials}} (X; A)^{\vec{m}}$$

- Generalised TPSA:

- TPSA extended to support parameters and non-uniform maximum truncation orders.

DA maps = array of multivariate real or complex Taylor polynomials (GTPSA).

- Monomials and Indexing:

$$\frac{\partial^3 f(x, y)}{\partial x^3}, \frac{\partial^3 f(x, y)}{\partial x^2 \partial y}, \frac{\partial^3 f(x, y)}{\partial x \partial y^2}, \text{ and } \frac{\partial^3 f(x, y)}{\partial y^3}$$

$$\partial x^3 = 3 \ 0, \ \partial x^2 \partial y = 2 \ 1, \ \partial x \partial y^2 = 1 \ 2, \ \text{and } \partial y^3 = 0 \ 3.$$

# Differential Algebra — Implementations

---

- **Finite differences DA:** Approximate accurately 1st order DA only.
  - Simple and fast, **manipulate values**, can be ported to GPUs, doesn't scale to high orders.
- **Symbolic DA:** Flexible and accurate but million times slower than other approaches.
  - Require a symbolic calculation engine (e.g. Mathematica).
- **Hand-coded DA (MAD-X):** Order is fixed and limited typically to 1st or 2nd order DA.
  - Error-prone, heavy to maintain, “easy” to adapt (e.g. add recipes).
- **Generated DA:** Order is fixed and limited typically from 1st to 4th order DA.
  - Require parametric polymorphism (complex) or code preprocessing (fragile).
- **Automatic Differentiation DA (MAD-NG):** Flexible and accurate, can scale to very high orders.
  - Require exact physics at all orders (complex), **manipulate functions** (complex), can be fast (complex).

# Differential Algebra — Representations

---

- Phase space **propagator**  $\mathcal{M}(z; s)z_0$  can be represented by **Matrix** and **Tensors** (MAD-X).
  - Such hard-coded representation results in *expansion, truncation and approximation*.
  - But it's easier and faster, yet heavy to maintain and extend (i.e. Matrix code).
- Phase space **propagator**  $\mathcal{M}(z; s)z_0$  can be represented by **Functions** application (MAD-NG).
  - Such dynamic representation has *no expansion, no truncation, and no approximation*.
  - But it's complex and slower when  $z_0$  is **not** a vector of scalars (i.e. DA code).

$$\begin{array}{lll}
 & \textbf{(Worst)} & \\
 \text{(Expanded)} & (\mathcal{M}_n \circ \dots \circ \mathcal{M}_2 \circ \mathcal{M}_1)(z_0) \neq \mathcal{M}_n(\dots(\mathcal{M}_2(\mathcal{M}_1(z_0)))\dots) & \text{(MAD-X)} \\
 \text{(Expanded knowing } z) & & \neq \widetilde{\mathcal{M}}_n(\dots(\widetilde{\mathcal{M}}_2(\widetilde{\mathcal{M}}_1(z_0)))\dots) \\
 \text{(Exact, Unified)} & & \neq \mathcal{F}_n(\dots(\mathcal{F}_2(\mathcal{F}_1(z_0)))\dots) \quad \text{(MAD-NG)} \\
 & & \textbf{(Best, no approximation)}
 \end{array}$$

where  $\mathcal{M}_n$  and  $\widetilde{\mathcal{M}}_n$  are propagators built respectively *a priori*, and *a posteriori* given  $z_{n-1}$ .

# Differential Algebra — Dual Representations

- Phase space **and propagator**  $\mathcal{M}(z; s)z_0$  can be represented by **Matrix** and **Tensors**, or by **Functions** and Taylor Polynomials using **AD**, named **TPSA** (Truncated Power Series Algebra).

- Only **TPSA** scale to high orders both for memory representation and speed of calculation.

- GTPSA (Generalised TPSA) are TPSA extended to support parameters and non-uniform maximum variables orders.

- GTPSA invented in 2014 and published at IPAC 2015.

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	168	504	1260	2772	5544	10296	18018	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

TPSA sizes

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	258	1554	9330	55986	335922	2015538	12093234	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Tensors sizes

# Differential Algebra — 2D Norm-L<sub>2</sub> Function Example

Example ( $f(x, y) = \sqrt{x^2 + y^2}$ )

```

gtpsad {nv=2, mo=3}           -- new descriptor (default)
local f = \x,y -> sqrt(x^2+y^2) -- the function f(x,y)
local x = tpsa() :setvar(2,1)  -- 1st variable x=2
local y = tpsa() :setvar(3,2)  -- 2nd variable y=3
f(x,y) :print "sqrt(x^2+y^2)"  -- f(x,y) |_(2,3)
    
```

- Outputs below are results from
  - $f(x, y) :deriv "01"$
  - $f(x, y) :integ "01"$

sqrt(x^2+y^2): R, NV = 2, MO = 3  
 \*\*\*\*\*

I	COEFFICIENT	ORDER	EXPONENTS
1	3.6055512754639891E+00	0	0 0
2	5.5470019622522915E-01	1	1 0
3	8.3205029433784372E-01	1	0 1
4	9.6005803192828115E-02	2	2 0
5	-1.2800773759043749E-01	2	1 1
6	4.2669245863479172E-02	2	0 2
7	-1.4770123568127402E-02	3	3 0
8	-2.4616872613545671E-03	3	2 1
9	2.2975747772642625E-02	3	1 2
10	-9.8467490454182682E-03	3	0 3

$$T_f^n(x; a) = \sum_{k=0}^n \frac{f_a^{(k)}}{k!} (x - a)^k.$$

$f(x, y)$	/0!
$x/f(x, y)$	/1!
$y/f(x, y)$	/1!
$y^2/f(x, y)^3$	/2!
$-xy/f(x, y)^3$	/1!/1!
$x^2/f(x, y)^3$	/2!
$-3xy^2/f(x, y)^5$	/3!
$y(2x^2 - y^2)/f(x, y)^5$	/2!/1!
$x(2y^2 - x^2)/f(x, y)^5$	/1!/2!
$-3x^2y/f(x, y)^5$	/3!

$$\frac{\partial f(x, y)}{\partial y}$$

$$\int f(x, y) dy$$

d sqrt(x^2+y^2) / dy: R, NV = 2, MO = 3  
 \*\*\*\*\*

I	COEFFICIENT	ORDER	EXPONENTS
1	8.3205029433784372E-01	0	0 0
2	-1.2800773759043749E-01	1	1 0
3	8.5338491726958343E-02	1	0 1
4	-2.4616872613545671E-03	2	2 0
5	4.5951495545285249E-02	2	1 1
6	-2.9540247136254805E-02	2	0 2

S sqrt(x^2+y^2) dy: R, NV = 2, MO = 3  
 \*\*\*\*\*

I	COEFFICIENT	ORDER	EXPONENTS
1	3.6055512754639891E+00	1	0 1
2	5.5470019622522915E-01	2	1 1
3	4.1602514716892186E-01	2	0 2
4	9.6005803192828115E-02	3	2 1
5	-6.4003868795218744E-02	3	1 2
6	1.4223081954493058E-02	3	0 3

( $\partial x, \partial y$ )

# Differential Algebra — 2D Gaussian Approximation Example

## Example (Modeling 2D Gaussian Surface)

```

gtpsad {nv=2, mo=5}
local sx, sy = 2, 3
local f = \x,y -> exp(-(x/sx)^2-(y/sy)^2)/(2*pi*sx*sy)
local x = tpsa() :setvar(0,1)      -- 1st variable
local y = tpsa() :setvar(0,2)      -- 2nd variable
local t = f(x,y) :print "N(0;{2,3})"
    
```

N(0;{2,3}): R, NV = 2, MO = 10

\*\*\*\*\*

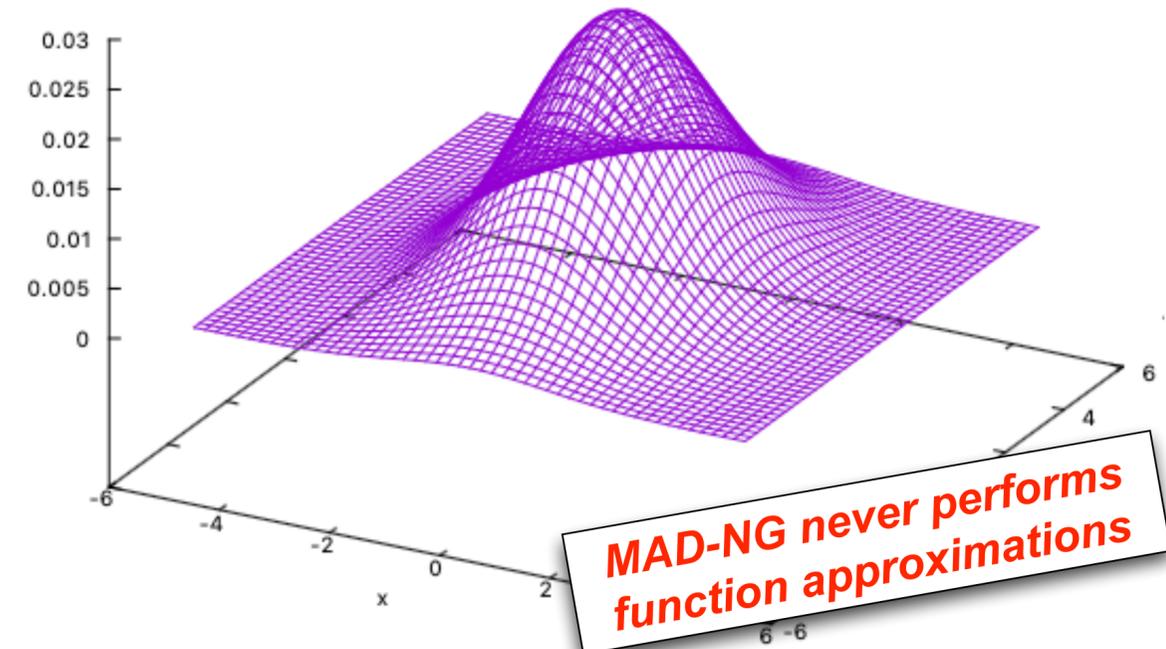
I	COEFFICIENT	ORDER	EXPONENTS				
1	2.6525823848649224E-02	0	0 0				
2	-6.6314559621623061E-03	2	2 0				
3	-2.9473137609610247E-03	2	0 2				
4	8.2893199527028826E-04	4	4 0				
5	7.3682844024025617E-04	4	2 2				
6	1.6373965338672359E-04	4	0 4	14	1.5161079017289220E-06	8	2 6
7	-6.9077666272524012E-05	6	6 0	15	1.6845643352543578E-07	8	0 8
8	-9.2103555030032021E-05	6	4 2	16	-2.1586770710163755E-07	10	10 0
9	-4.0934913346680898E-05	6	2 4	17	-4.7970601578141674E-07	10	8 2
10	-6.0644316069156879E-06	6	0 6	18	-4.2640534736125931E-07	10	6 4
11	4.3173541420327508E-06	8	8 0	19	-1.8951348771611525E-07	10	4 6
12	7.6752962525026679E-06	8	6 2	20	-4.2114108381358946E-08	10	2 8
13	5.1168641683351122E-06	8	4 4	21	-3.7434763005652394E-09	10	0 10

(∂x, ∂y)

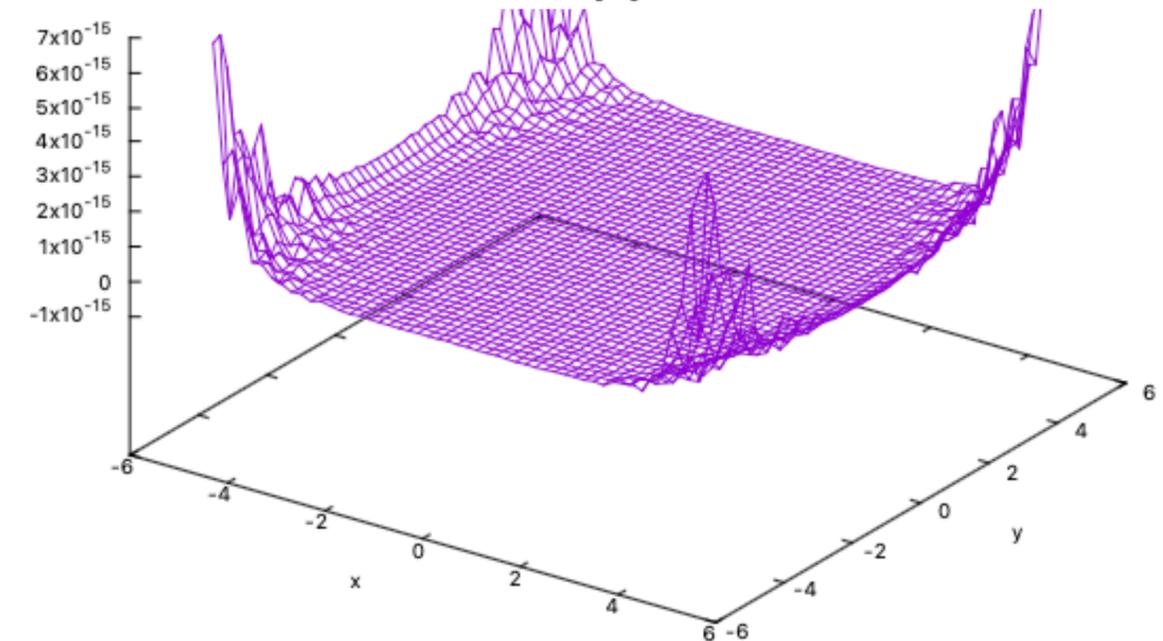
(∂x, ∂y)

- Plots aside result from

- mo=100
- t:eval{x,y} (top graph)
- t:eval{x,y}-f(x,y)



## Gaussian Approximation



Approximation Error ( $\approx 10^{-15}$ ) 10

# Concrete Example — Exact vs Expanded Drift

- Hamiltonian of an exact straight drift (no bend, no field) with  $z = (x, p_x, y, p_y, \delta; s)$ .

$$H = -\sqrt{(1 + \delta)^2 - p_x^2 - p_y^2} \stackrel{\text{def}}{=} -p_z \quad ; \quad p_x = \frac{P_x}{P_0}, \quad p_y = \frac{P_y}{P_0}, \quad \delta = \frac{P - P_0}{P_0}$$

- Hamilton's equations

$$\frac{dx}{ds} = \{x, H\} = \frac{\partial H}{\partial p_x} = \frac{p_x}{p_z} \quad ; \quad \frac{dy}{ds} = \{y, H\} = \frac{\partial H}{\partial p_y} = \frac{p_y}{p_z} \quad ; \quad \frac{dp_{x,y}}{ds} = -\frac{\partial H}{\partial x,y} = 0$$

- Phase space propagator through an **exact** straight drift of length  $L$  (TPSA code, MAD-NG)

$$z(s) = e^{\{., LH\}} z(s_0) = z + \{z, LH\} + \frac{L^2}{2!} \underbrace{\{\{z, H\}, H\}}_{=0} + \dots \implies \begin{cases} x(s) = x(s_0) + L \frac{p_x}{p_z} \\ y(s) = y(s_0) + L \frac{p_y}{p_z} \end{cases}$$

- Expanded** solution (Matrix code, MAD-X)

$$\frac{p_x}{p_z} = \underbrace{\frac{p_x}{1 + \delta}}_{\text{linear}} + \underbrace{\frac{p_x(p_x^2 + p_y^2)}{2(1 + \delta)^3}}_{\text{cubic}} + \dots \implies \begin{cases} x(s) = x(s_0) + \frac{L}{1 + \delta} p_x \\ y(s) = y(s_0) + \frac{L}{1 + \delta} p_y \end{cases} \quad M_{\text{drift}} = \begin{pmatrix} 1 & \frac{L}{1 + \delta} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{L}{1 + \delta} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Concrete Example — Tracking Through a Drift

- Code for the exact straight drift.  
(physics provided by MAD-NG)

$$\implies \begin{cases} x(s) = x(s_0) + L \frac{p_x}{p_z} \\ y(s) = y(s_0) + L \frac{p_y}{p_z} \end{cases}$$

```
for i=1,m.npar do  -- Loop over N particles or N damaps
  local x, px, y, py, pt in m[i]
  local l_pz = 1 / sqrt(1 + 2/beta*pt + pt^2 - px^2 - py^2)
  m[i].x = x + px*l_pz
  m[i].y = y + py*l_pz
end
```

$= (1 + \delta)^2$

- Code that builds a sequence containing a single drift of  $L = 2m$ .
- Code for tracking a single particle (vector of scalars) through a drift.
- Code for tracking a single DA map of order 2 through a drift.

```
local seq = sequence { beam=beam,  -- Attach default beam
  drift 'dft' { l=2 },
}

local x0 = { px=1e-3, py=1e-4 }  -- Phase space (particle)

local _, mflw = track { sequence=seq, x0=x0 }
print(tostring(mflw[1]))  -- Phase space at the end (particle)

local _, mflw = track { sequence=seq, x0=x0, mapdef=2 }
mflw[1]:print("mflw")  -- Phase space at the end (DA map)
```

# Concrete Example — Tracking Through a Drift (output)

- Output of the tracking of x0 at order 2.

- Expanded drift map at order 3 (x0 = id).

```
[6R] mflw (x0) =
X      : R, NV = 6, MO = 2
*****
I      COEFFICIENT          ORDER  EXPONENTS
1      2.0000010100007654E-03    0      0 0 0 0
2      1.0000000000000000E+00    1      1 0 0 0
3      2.0000030100037951E+00    1      0 1 0 0
4      2.0000030300038257E-07    1      0 0 0 1
5      -2.0000032911241996E-03   1      0 0 0 0
6      3.0000075450133134E-03    2      0 2 0 0
7      2.0000090300189758E-04    2      0 1 0 1
8      1.0000015450019886E-03    2      0 0 0 2
9      -2.0000092911401328E+00   2      0 1 0 0
10     -6.0000159333886912E-07   2      0 0 0 1
11     2.0000068433734400E-03    2      0 0 0 0
*****
Y      : R, NV = 6, MO = 2
*****
I      COEFFICIENT          ORDER  EXPONENTS
1      2.0000010100007653E-04    0      0 0 0 0
2      2.0000030300038257E-07    1      0 1 0 0
3      1.0000000000000000E+00    1      0 0 1 0
4      2.0000010300007958E+00    1      0 0 0 1
5      -2.0000032911241994E-04   1      0 0 0 0
6      1.0000045150094879E-04    2      0 2 0 0
7      2.0000030900039771E-03    2      0 1 0 1
*****
      (cont.)
```

$\partial.(x, p_x, y, p_y, t, p_t)$

```
[6R] mflw (drift transfer map) =
X      : R, NV = 6, MO = 3
*****
I      COEFFICIENT          ORDER  EXPONENTS
1      1.0000000000000000E+00    1      1 0 0 0
2      2.0000000000000000E+00    1      0 1 0 0
3      -2.0000002611199781E+00   2      0 1 0 0
4      1.0000000000000000E+00    3      0 3 0 0
5      1.0000000000000000E+00    3      0 1 0 2
6      2.0000007833599858E+00    3      0 1 0 0
*****
Y      : R, NV = 6, MO = 3
*****
I      COEFFICIENT          ORDER  EXPONENTS
1      1.0000000000000000E+00    1      0 0 1 0
2      2.0000000000000000E+00    1      0 0 0 1
3      -2.0000002611199781E+00   2      0 0 0 1
4      1.0000000000000000E+00    3      0 2 0 1
5      1.0000000000000000E+00    3      0 0 0 3
6      2.0000007833599858E+00    3      0 0 0 1
*****
      (cont.)
```

$$\frac{p_x}{p_z} = \frac{p_x}{1 + \delta} + \frac{p_x(p_x^2 + p_y^2)}{2(1 + \delta)^3}$$

$\partial.(x, p_x, y, p_y, t, p_t)$

$\mathcal{M}(z; s)z_0$  (map composition)

# Concrete Example — Exact vs Expanded Drift (output)

- Tracking through an exact drift (MAD-NG).
- Tracking through an expanded drift (MAD-X).

```
[6R] mflw (x0) =
```

X : R, NV = 6, MO = 2

```
*****
```

I	COEFFICIENT	ORDER	EXPONENTS
1	2.0000010100007654E-03	0	0 0 0 0 0 0
2	1.0000000000000000E+00	1	1 0 0 0 0 0
3	2.0000030100037951E+00	1	0 1 0 0 0 0
4	2.0000030300038257E-07	1	0 0 0 1 0 0
5	-2.0000032911241996E-03	1	0 0 0 0 0 1
6	3.0000075450133134E-03	2	0 2 0 0 0 0
7	2.0000090300189758E-04	2	0 1 0 1 0 0
8	1.0000015450019886E-03	2	0 0 0 2 0 0
9	-2.0000092911401328E+00	2	0 1 0 0 0 1
10	-6.0000159333886912E-07	2	0 0 0 1 0 1
11	2.0000068433734400E-03	2	0 0 0 0 0 2

*∂.(x, p<sub>x</sub>, y, p<sub>y</sub>, t, p<sub>t</sub>)*

Y : R, NV = 6, MO = 2

```
*****
```

I	COEFFICIENT	ORDER	EXPONENTS
1	2.0000010100007653E-04	0	0 0 0 0 0 0
2	2.0000030300038257E-07	1	0 1 0 0 0 0
3	1.0000000000000000E+00	1	0 0 1 0 0 0
4	2.0000010300007958E+00	1	0 0 0 1 0 0
5	-2.0000032911241994E-04	1	0 0 0 0 0 1
6	1.0000045150094879E-04	2	0 2 0 0 0 0
7	2.0000030900039771E-03	2	0 1 0 1 0 0 (cont.)

```
[6R] mflw (id) * z0 =
```

X : R, NV = 6, MO = 3

```
*****
```

I	COEFFICIENT	ORDER	EXPONENTS
1	2.0000010099999999E-03	0	0 0 0 0 0 0
2	1.0000000000000000E+00	1	1 0 0 0 0 0
3	2.0000030099999999E+00	1	0 1 0 0 0 0
4	2.0000000000000002E-07	1	0 0 0 1 0 0
5	-2.0000002611199781E-03	1	0 0 0 0 0 1
6	3.0000000000000001E-03	2	0 2 0 0 0 0
7	2.0000000000000001E-04	2	0 1 0 1 0 0
8	1.0000000000000000E-03	2	0 0 0 2 0 0
9	-2.0000002611199781E+00	2	0 1 0 0 0 1
10	2.0000007833599859E-03	2	0 0 0 0 0 2 (cont.)

*∂.(x, p<sub>x</sub>, y, p<sub>y</sub>, t, p<sub>t</sub>)*

Y : R, NV = 6, MO = 3

```
*****
```

I	COEFFICIENT	ORDER	EXPONENTS
1	2.0000010100000001E-04	0	0 0 0 0 0 0
2	2.0000000000000002E-07	1	0 1 0 0 0 0
3	1.0000000000000000E+00	1	0 0 1 0 0 0
4	2.0000010300000000E+00	1	0 0 0 1 0 0
5	-2.0000002611199783E-04	1	0 0 0 0 0 1
6	1.0000000000000000E-04	2	0 2 0 0 0 0
7	2.0000000000000000E-03	2	0 1 0 1 0 0 (cont.)

# Concrete Example — Exact Drift using Lie Operators

- Code using various Lie methods to calculate the map  $\mathcal{M}(z; s)z_0$  resulting from the tracking of  $x_0$  through an exact straight drift of length  $L$ .

- Formula used from previous slides

$$H = -\sqrt{(1 + \delta)^2 - p_x^2 - p_y^2}$$

$$z(s) = e^{\{\cdot, H\}} z(s_0) = \mathcal{M}(z; s)z(s_0)$$

$$z(s) = e^{\{\cdot, LH\}} z(s_0) = z + \{z, LH\}$$

$$e^{\{\cdot, H\}} z = z + \{z, H\} + \frac{1}{2!} \{\{z, H\}, H\} + \dots$$

- All the outputs are strictly equal to  $\mathcal{M}(z; s)z_0$  within e.g. **1e-15** for all orders.

```

local z0 = damap{ nv=6, mo=3 }:set0(x0) -- set orbit
local L = seq.dft.l
local px, py, pt in z0 -- px,py,pt are Taylor Polynomials

-- Hamiltonian of the exact straight drift
local H = -sqrt(1 + 2/beta*pt + pt^2 - px^2 - py^2)

-- Explicit method for each variables
local z = z0:same()
z.x = z0.x + z0.x:poisbra(L*H) -- x + {x,LH}
z.px = z0.px
z.y = z0.y + z0.y:poisbra(L*H) -- y + {y,LH}
z.py = z0.py

-- Explicit propagator
local F = -z0:vec2fld(L*H) -- -{LH,.} = {.,LH}
local zp = z0 + F -- z + {z,LH}

-- Full Lie propagator
local zpp = z0:exppb(F) -- exp({.,LH})z

H:print("H") -- All z output are equal to track mflw(x0)
z:print("z") ; zp:print("z'") ; zpp:print("z''")

```

# Concrete Example — Thick Sextupole (Track & Lie)

- Code for the exact thick sextupole.  
(physics provided by MAD-NG)

$$H = H_{\text{drift}} + \frac{k_2}{6}(x^3 - 3xy^2)$$

$$z(s) = e^{\{\cdot, H\}} z(s_0) = \mathcal{M}(z; s) z(s_0)$$

$$\frac{dp_x}{ds} = - \frac{\partial H}{\partial x}$$

```
[6R] sf transfer map =
X      :  R, NV =  6, MO =  2
*****
I  COEFFICIENT      ORDER  EXPONENTS
1  1.0000000000000000E+00  1    1 0 0 0 0
2  5.0000000000000000E-01  1    0 1 0 0 0
3 -6.2500000000000000E-01  2    2 0 0 0 0
4 -2.0833333333333323E-01  2    1 1 0 0 0
5 -2.6041666666666657E-02  2    0 2 0 0 0
6  6.2500000000000000E-01  2    0 0 2 0 0
7  2.0833333333333323E-01  2    0 0 1 1 0
8  2.6041666666666657E-02  2    0 0 0 2 0
9 -5.0000006527999452E-01  2    0 1 0 0 0 1
*****
PX     :  R, NV =  6, MO =  2
*****
I  COEFFICIENT      ORDER  EXPONENTS
1  1.0000000000000000E+00  1    0 1 0 0 0
2 -2.5000000000000000E+00  2    2 0 0 0 0
3 -1.2500000000000000E+00  2    1 1 0 0 0
4 -2.0833333333333337E-01  2    0 2 0 0 0
5  2.5000000000000000E+00  2    0 0 2 0 0
6  1.2500000000000000E+00  2    0 0 1 1 0
7  2.0833333333333337E-01  2    0 0 0 2 0
```

$\partial.(x, p_x, y, p_y, t, p_t)$

```
local seq = sequence { beam=beam,
  sextupole 'sf' { l=0.5, k2=10 },
}

local _, mflw = track { sequence=seq, mapdef=2 }
mflw[1]:print("sf transfer map")

-- Lie methods -----
local z0 = damap{ nv=6, mo=3 }
local l, k2 in seq.sf
local x, px, y, py, pt in z0 -- Taylor Polynomials

-- Hamiltonian of the thick sextupole
local H = H_drift + k2/6 * (x^3 - 3*x*y^2)

-- Full Lie propagator
local z = z0:exppb(l*H) -- exp({.,LH})z0

H:print("H") -- Output: z equals track's mflw(x0)
z:cutord(3):print("z")
```

- Output of *mflw* and *z* are strictly equal within e.g. **1e-15** for all orders.

# Parametric Phase Space — Thick Sextupole (Track & Lie)

- Code for the exact thick sextupole.  
(physics provided by MAD-NG)

$$H = H_{\text{drift}} + \frac{k_2}{6}(x^3 - 3xy^2)$$

$$z(s; k_2) = e^{\{\cdot, H\}} z(s_0; k_2) = \mathcal{M}(z; s) z(s_0; k_2)$$

```
[6R] sf parametric transfer map =
X      : R, NV = 6, MO = 3, NP = 1, PO = 1
*****
I      COEFFICIENT          ORDER  EXPONENTS
1      1.0000000000000000E+00    1      1 0 0 0 0
2      5.0000000000000000E-01    1      0 1 0 0 0
3      -6.2500000000000000E-01    2      2 0 0 0 0
4      -2.0833333333333323E-01    2      1 1 0 0 0
5      -2.6041666666666657E-02    2      0 2 0 0 0
6      6.2500000000000000E-01    2      0 0 2 0 0
7      2.0833333333333323E-01    2      0 0 1 1 0
8      2.6041666666666657E-02    2      0 0 0 2 0
9      -5.0000006527999452E-01    2      0 1 0 0 0
      . . . (3 order in variables only are omitted)
27     -6.2500000000000000E-02    3      2 0 0 0 0 7^1
28     -2.0833333333333322E-02    3      1 1 0 0 0 7^1
29     -2.6041666666666661E-03    3      0 2 0 0 0 7^1
30      6.2500000000000000E-02    3      0 0 2 0 0 7^1
31      2.0833333333333322E-02    3      0 0 1 1 0 7^1
32      2.6041666666666661E-03    3      0 0 0 2 0 7^1
```

$$\partial.(x, p_x, y, p_y, t, p_t; k_2)$$

```
local k2, l in seq.sf -- k2=10, l=0.5
local z0 = damap{nv=6, mo=3, np=1, po=1, pn={"k2"}}
seq.sf.k2 = k2 + z0.k2

local _, mflw = track { sequence=seq, x0=z0 }
mflw[1]:print("sf parametric transfer map")

-- Lie methods - - - - -
local z0 = damap{nv=6, mo=4, np=1, po=1, pn={"k2"}}
local x, px, y, py, pt in z0 -- Taylor Polynomials
k2 = k2 + z0.k2

-- Hamiltonian of the thick sextupole
local H = H_drift + k2/6 * (x^3 - 3*x*y^2)

-- Full Lie propagator
local z = z0:exppb(l*H) -- exp({., LH})z0

H:print("H") -- Output: z equals track's mflw(z0)
z:cutord(4):print("z")
```

- Output of *mflw* and *z* are strictly equal within e.g. **1e-15** for all orders.

# Parametric Phase Space — Sextupole Sensitivity Map

- Code for the exact thick sextupole.  
(physics provided by MAD-NG)

$$H = H_{\text{drift}} + \frac{k_2}{6}(x^3 - 3xy^2)$$

$$z(s; k_2) = e^{\{\cdot, H\}} z(s_0; k_2) = \mathcal{M}(z; s) z(s_0; k_2)$$

```
local z0 = damap{nv=6,mo=5,np=1,po=2,pn={"k2"}}
seq.sf.k2 = z0.k2 -- Switch off the sextupole!!!

-- Track through a sextupole turned off!!!
local _, mflw = track { sequence=seq, x0=z0 }
mflw[1]:print("sf parametric sensitivity map")
```

$k_2 = 0$

- Parametric maps are extremely powerful for studying lattice sensitivities and optimising quantities, as they provide the *exact Jacobian* to the Match command and offer a way to explore behaviour around a chosen setup without risking instability.

[6R] sf parametric sensitivity map =

PX : R, NV = 6, MO = 5, NP = 1, PO = 2  
\*\*\*\*\*

I	COEFFICIENT	ORDER	EXPONENTS
1	1.0000000000000000E+00	1	0 1 0 0 0 0
2	-2.5000000000000000E-01	3	2 0 0 0 0 0
3	-1.2500000000000000E-01	3	1 1 0 0 0 0
4	-2.0833333333333339E-02	3	0 2 0 0 0 0
5	2.5000000000000000E-01	3	0 0 2 0 0 0
6	1.2500000000000000E-01	3	0 0 1 1 0 0
7	2.0833333333333339E-02	3	0 0 0 2 0 0
8	1.2500001631999863E-01	4	1 1 0 0 0 1
9	4.1666672106666219E-02	4	0 2 0 0 0 1
10	-1.2500001631999863E-01	4	0 0 1 1 0 1
11	-4.1666672106666219E-02	4	0 0 0 2 0 1
12	-6.2500000000000000E-02	5	1 3 0 0 0 0
13	-2.0833333333333339E-02	5	0 4 0 0 0 0
14	6.2500000000000000E-02	5	0 2 1 1 0 0
15	-6.2500000000000000E-02	5	1 1 0 2 0 0
16	6.2500000000000000E-02	5	0 0 1 3 0 0
17	2.0833333333333339E-02	5	0 0 0 4 0 0
18	-1.2500004895999911E-01	5	1 1 0 0 0 2
19	-6.2500021759999622E-02	5	0 2 0 0 0 2
20	1.2500004895999911E-01	5	0 0 1 1 0 2
21	6.2500021759999622E-02	5	0 0 0 2 0 2
22	1.0416666666666675E-02	5	3 0 0 0 0 0
23	6.5104166666666748E-03	5	2 1 0 0 0 0
24	1.8056665177785579E-03	5	1 2 0 0 0 0
25	2.3440274055575039E-04	5	0 3 0 0 0 0
26	1.0416666666666675E-02	5	1 0 2 0 0 0
27	-1.3020833333333374E-03	5	0 1 2 0 0 0
28	7.8125000000000035E-03	5	1 0 1 1 0 0
29	-2.2865051559873423E-04	5	0 1 1 1 0 0
30	2.0343170333772934E-03	5	1 0 0 2 0 0
31	2.3440274055575039E-04	5	0 1 0 2 0 0

X omitted as it is also affected by the exact drift (too long).

$\frac{\partial^3 p_x}{\partial x^2 \partial k_2}$

$\frac{\partial^4 p_x}{\partial x \partial p_x \partial p_t \partial k_2}$

$\frac{\partial^5 p_x}{\partial y \partial p_y \partial p_t^2 \partial k_2}$

$\frac{\partial^5 p_x}{\partial x \partial y \partial p_y \partial k_2^2}$

$\partial.(x, p_x, y, p_y, t, p_t; k_2)$



# Nonlinear Optics — Octupolar RDTs vs MS for HL-LHC

- Sensitivity of HL-LHC optics to octupolar RDTs vs MS in Arc 45.

$$f_{jklm} = \frac{h_{jklm}}{1 - e^{2\pi i[(j-k)\nu_x + (l-m)\nu_y]}}$$

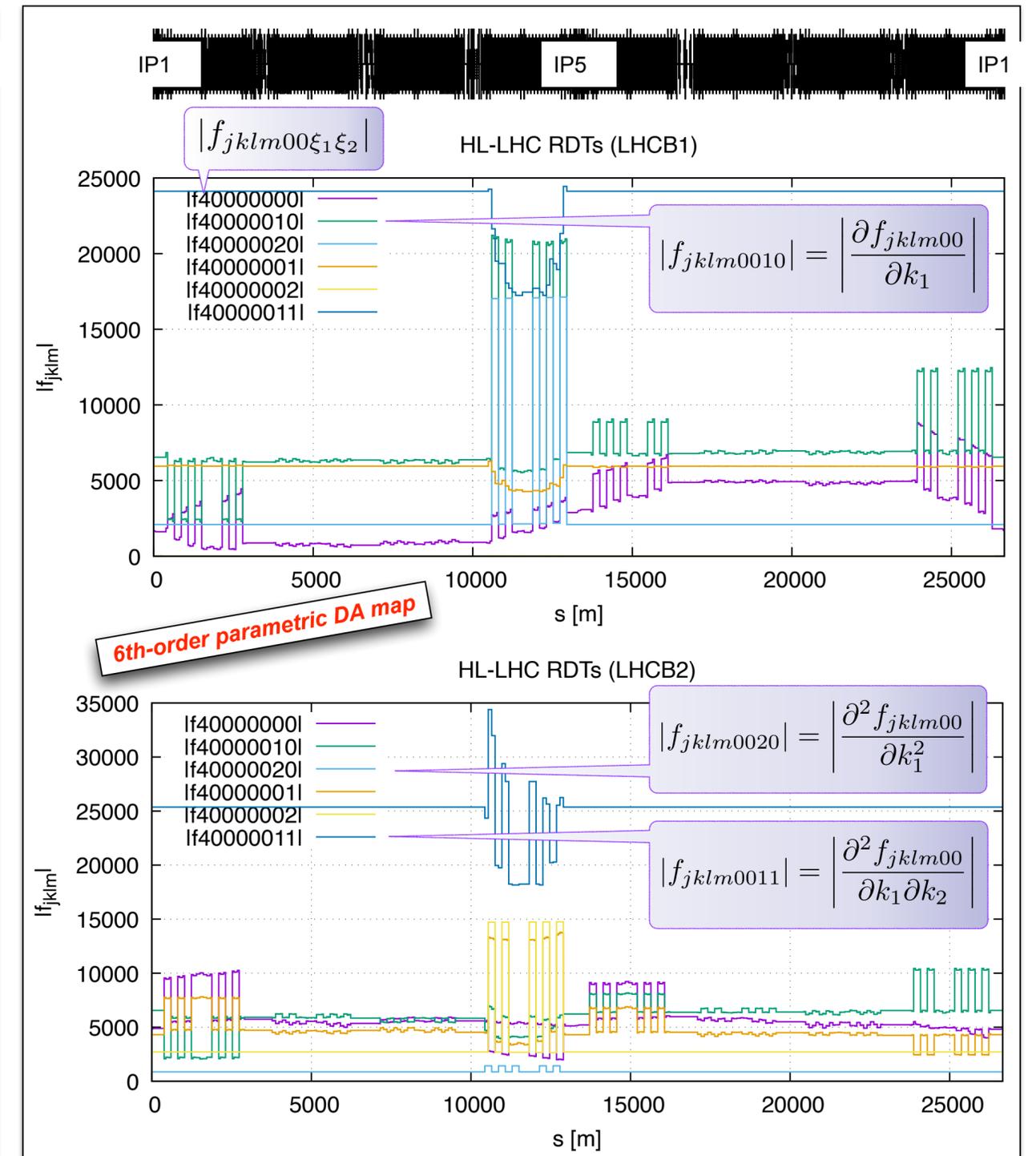
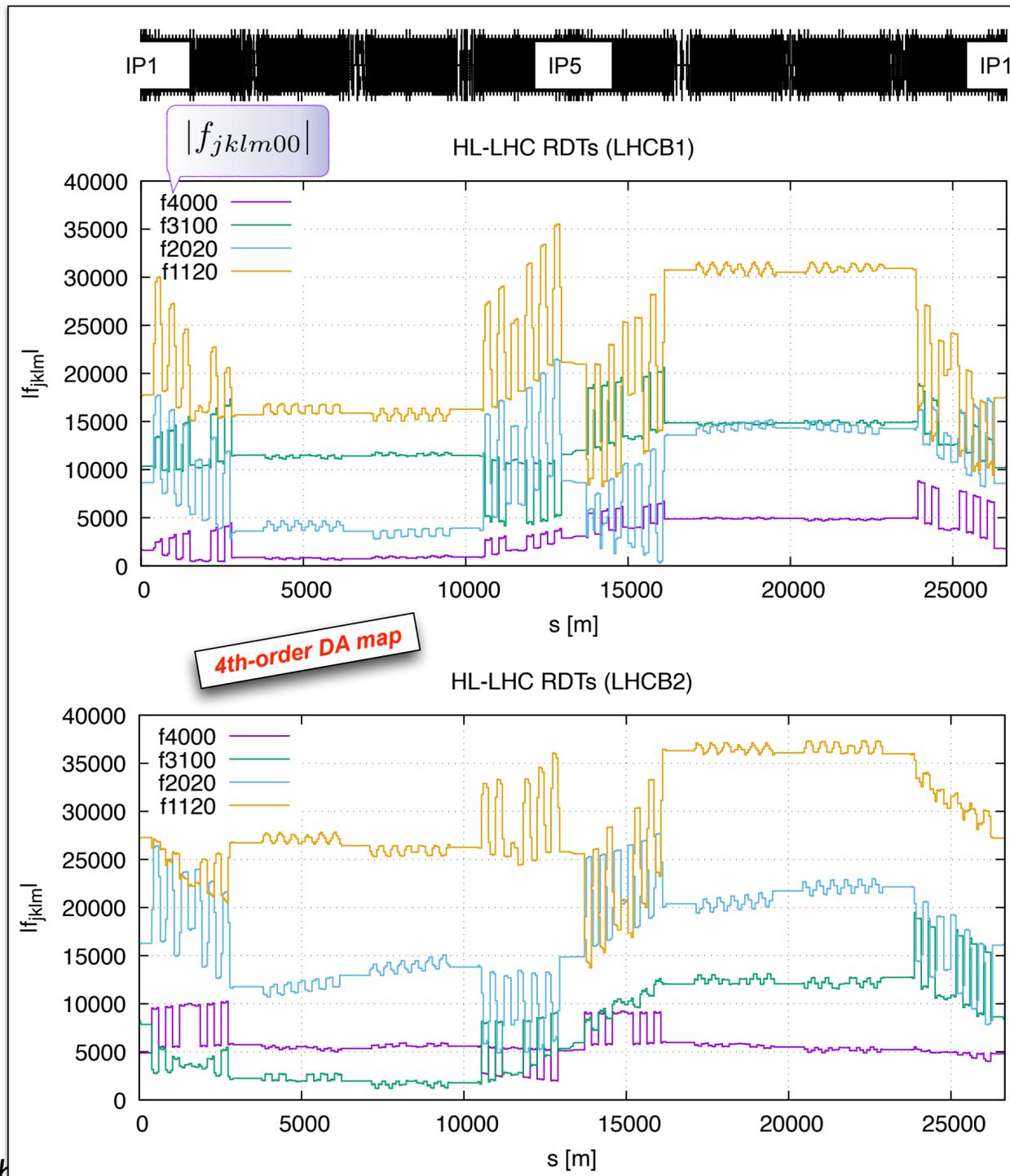
Resonances:  $N = (j - k)Q_x + (l - m)Q_y$

Spectral lines:  $H(1 + k - j, m - l)$   
 $V(k - j, 1 + m - l)$

$$f_{jklmno\xi_1\xi_2} = \frac{\partial^{\xi_1 + \xi_2} f_{jklmno}}{\partial k_1^{\xi_1} \partial k_2^{\xi_2}}$$

DA map order =  $j + k + l + m + n + o + \sum_i \xi_i$

These 2 RDTs' plots take  
 32 sec in MAD-NG  
 40 min in MADX-PTC

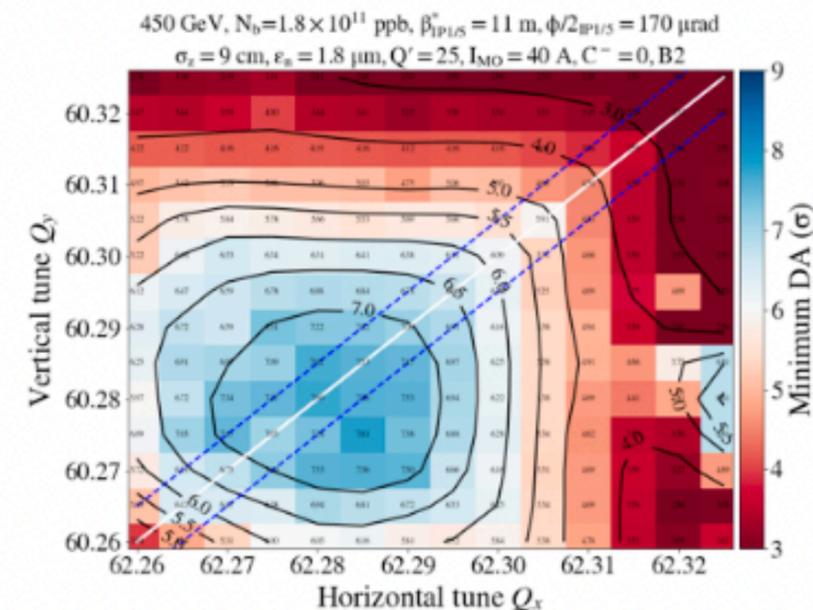
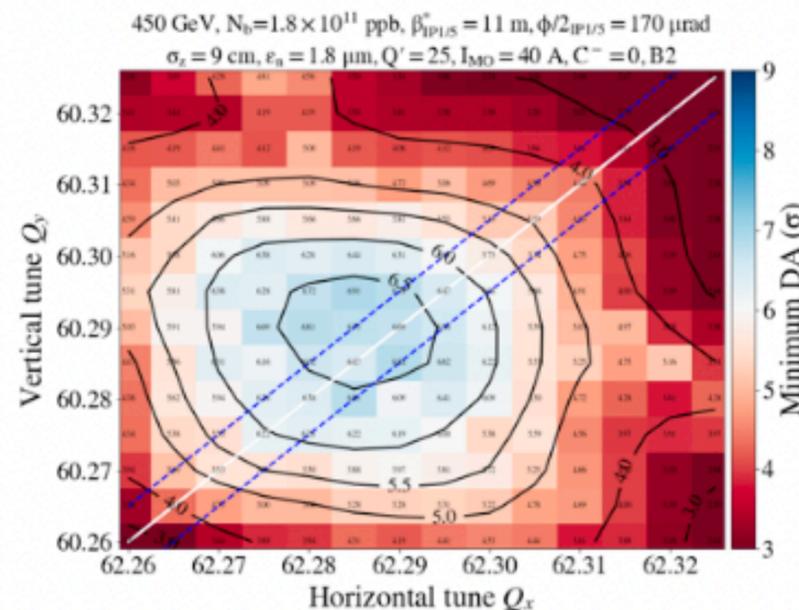
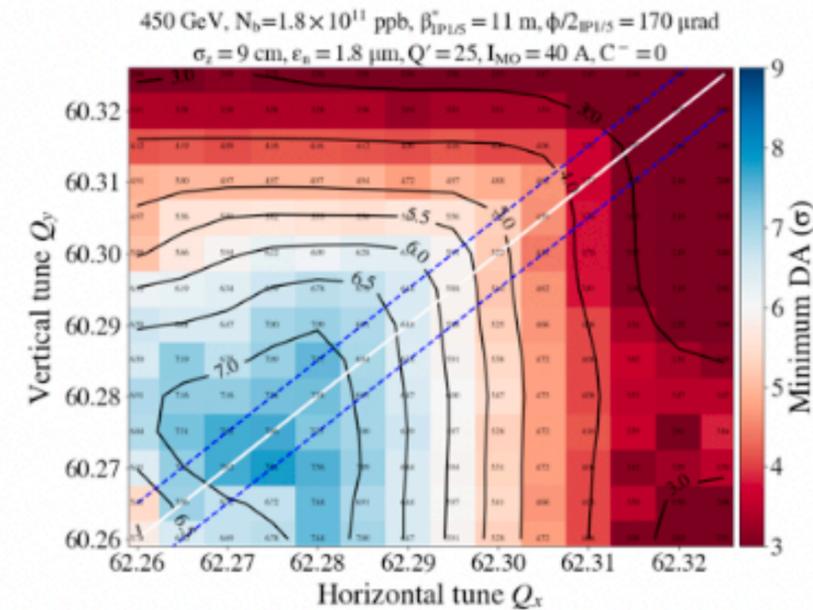
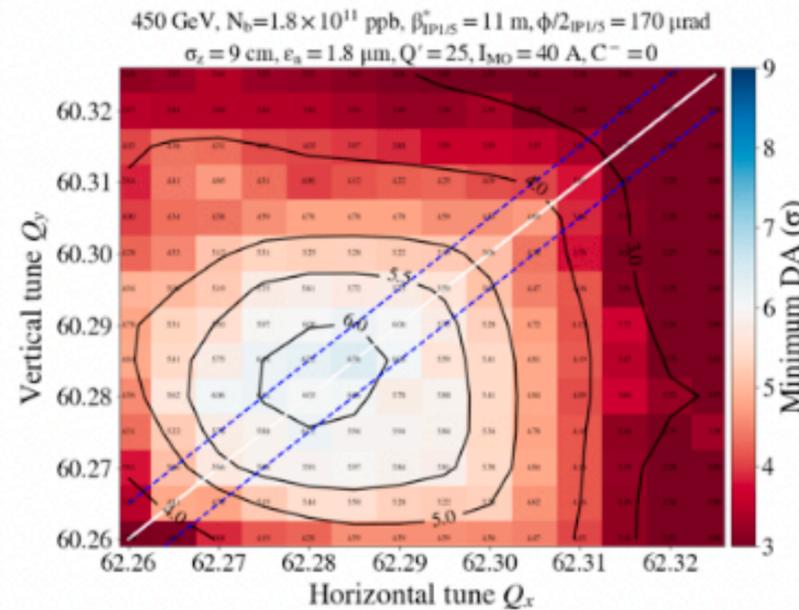


# Nonlinear Optics — Lowering RDTs for LHC@injection

## Dynamic Aperture Improvements

- Lowering **Octupolar RD** MO knobs) and optimis (see also next slide)

Dynamic aperture for beam 1 (top) and beam 2 (bottom) with old (left) and new (right) injection optics for LHC. Lowering the octupolar RDTs has significantly improved the dynamic aperture at injection.



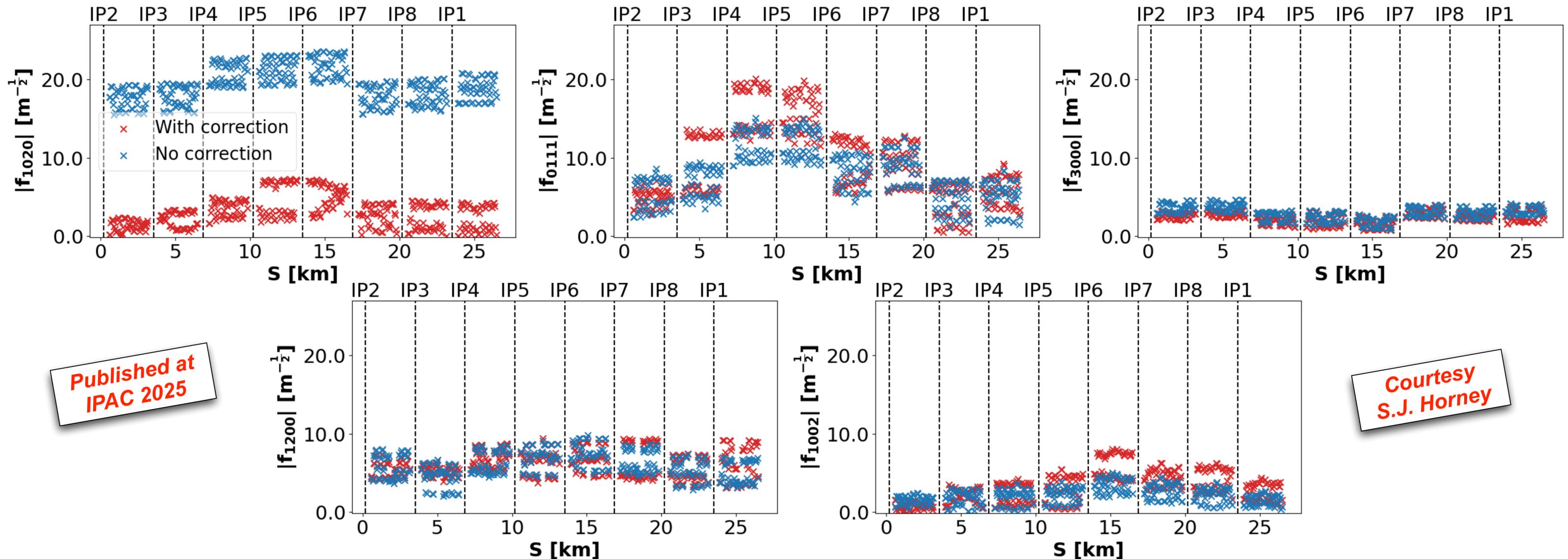
Courtesy S. Kostoglou

Beam lifetime x3 @ injection in 2023

Published at HB 2023

# Nonlinear Optics — Lowering $f_{1020}$ RDT for LHC B3

- Lowering **Sextupolar RDT**  $f_{1020}$  at injection and during the ramp using MAD-NG parametric DA maps (MS knobs) and optimiser without degrading other sextupolar RDTs.



# Architecture and Performance

---

- LuaJIT: high-performance scripting engine <https://indico.cern.ch/event/487416/contributions/2174904/>
  - Tracing JIT compiler delivering C-like execution speed, completely transparent to the user with no warm-up delay.
  - Zero-cost abstractions: user scripts run at compiled speed while retaining full access to all underlying physics.
  - Enables fast prototyping without sacrificing performance (e.g. new physics).
  - Provides simple and fast interfacing with external libraries (e.g. zero overhead for GTPSA).
- Object-oriented, lightweight and powerful data model
  - Accelerator objects implemented with minimal memory footprint.
  - Fast attribute access, internal caching, and efficient scanning of object.
  - Inheritance improves scaling to very large and complex lattices (i.e. everything is an object).
- Deferred expressions (symbolic-on-demand evaluation)
  - User expressions (functions) stored symbolically and evaluated only when needed (e.g. qf.k1: expr. or value?).
  - Enables various optics configurations, parametric studies and sensitivity analysis without modifying the lattice.
  - Eliminates redundant computations and manual bookkeeping — thanks to the JIT.

# Physics Performance — Accuracy First

- MAD-NG always includes state-of-the-art physics, no compromise, yet remains much faster than PTC.
- MAD-NG opens new possibilities, being fast enough to train modern ML models.
- Lua vs. C++ code

Exact drift in curved frame

```
local function curex_drift (elm, m, lw, istp)

    local ld = (m.eld or m.el)*lw
    local ang, rho = m.eh*m.el*lw*m.edir, 1/m.eh*m.edir
    local ca, sa, sa2 = cos(ang), sin(ang), sin(ang/2)
    local beta in m.beam

    for i=1,m.npar do
        local x, px, y, py, t, pt in m[i]

        local dpp1 = 1 + 2/beta*pt + pt^2
        local pz = sqrt(dpp1 - px^2 - py^2)
        local _pz = 1/pz
        local pxt = px*_pz
        local _ptt = 1/(ca - sa*pxt)
        local pst = (x+rho)*sa*_pz*_ptt

        m[i].x = (x + rho*(2*sa2^2 + sa*pxt))*_ptt
        m[i].px = ca*px + sa*pz
        m[i].y = y + pst*py
        m[i].t = t - pst*(1/beta+pt) + (1-m.T)/beta*ld
    end
end
```

Lua code

```
template <typename M,          - type of map flow
         typename T=M::T,    - type of variable
         typename P=M::P,    - type of parameter
         typename R=M::R>    - type of parameter reference
inline void curex_drift (cflw<M> &m, num_t lw, int istp)
{
    P ld = (fval(m.eld) ? R(m.eld) : R(m.el))*lw;
    P ang = R(m.eh)*R(m.el)*lw*m.edir, rho = 1/R(m.eh)*m.edir;
    P ca = cos(ang), sa = sin(ang), sa2 = sin(ang/2);

    FOR(i,m.npar) {
        M p(m,i);

        T dpp1 = 1 + 2/m.beta*p.pt + sqr(p.pt);
        T pz = sqrt(dpp1 - sqr(p.px) - sqr(p.py));
        T _pz = 1/pz;
        T pxt = p.px*_pz;
        T _ptt = 1/(ca - sa*pxt);
        T pst = (p.x+rho)*sa*_pz*_ptt;

        p.x = (p.x + rho*(2*sqr(sa2) + sa*pxt))*_ptt;
        p.px = ca*p.px + sa*pz;
        p.y += pst*p.py;
        p.t -= pst*(1/m.beta+p.pt) - (1-m.T)/m.beta*ld;
    }
}
```

C++ code

- Both are simple to read: extensive use of **operator overloading**.
- Both are **fully polymorphic**: same code for particles, DA maps, and parametric DA maps.
- Performances: GTPSA in C + Fast memory allocator + C++ advanced techniques lead to better **memory management** than Lua's garbage collector, resulting in a **speed improvement of x7**.

# Summary

---

- ***GTPSA and Automatic Differentiation.***

- GTPSA are objects representing high order multivariate real or complex Taylor polynomials. GTPSA use forward automatic differentiation to calculate exact derivatives and functions with an ***accuracy close to symbolic calculations*** (relative error within few  $\epsilon=2.2e-16$ ).

- ***Tracking and Lie algebra.***

- MAD-NG provides a fast tracking engine with exact, thick-element physics and accurate models, together with Lie-algebra tools to work directly with the Hamiltonian and the DA maps (vector field, PB, Lie exp, Lie log, Dragt-Finn factorisation, +500 functions, etc.).

- ***Parametric DA maps.***

- Phase space DA maps can be parameterised by users with any lattice quantities that influence its tracking (strengths, lengths, positions, misalignments, etc.), enabling the study of ***optics sensitivities*** around a working point without modifying the lattice (thanks to deferred expressions), and thus avoiding the risk of moving close to instabilities.

# Backup Slides

# A Bit of History

---

- 1996-1998 **SL++**: scientific library in C++, full of template expressions, including AD.
- 2011-2022 Leading the **MAD-X team**, improving the application to meet CERN's needs.
- 2014-2015 Development of the new **Differential Algebra** core (**GTPSA**, IPAC 2015).
- 2016-2018 Development of **MAD-NG** for **linear optics** as a replacement for MAD-X.
  - First Twiss calculations on LHC and HL-LHC.
- 2019-2021 Major improvements on many aspects: features, physics, accuracy, and speed.
  - New optimiser (20+ algorithms), support of **parametric DA maps**.
- 2022-2024 Extension of **MAD-NG** to **nonlinear optics** as a replacement for MADX-PTC.
  - General use of high-order parametric differential maps, version 1.0 released.

# Differential Algebra — Finite Diff. vs Auto. Diff.

- Taylor Polynomial:  $T_f^n(x; a) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \dots + \frac{f^{(n)}(a)}{n!}(x - a)^n$   
(Functions)
- Finite Differences:  $f(x + h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \dots + \frac{h^n}{n!}f^{(n)}(x)$   
(Scalars)
- Finite Differences approximations don't work for **high-orders** or **multivariate** functions or "near" **singularities** (e.g. abs, inv, sqrt, log, resonance, ...).

$f(x) = x^{-1}$	FD $_{h=10^{-4}}$	TP
$f(10^{-2})^{(1)} = -10^5$	-10 001	$-10^5$
$f(10^{-5})^{(1)} = 10^{10}$	$\approx 10^8$	$10^{10}$

$f(x) =  x $	FD $_{h=10^{-4}}$	TP
$f(10^{-8})^{(1)} = 1$	$1.0000e - 04$	1
$f(10^{-8})^{(2)} = 0$	$1.9998e + 04$	0
$f(10^{-8})^{(3)} = 0$	$-1.0000e + 04$	0
$f(10^{-8})^{(4)} = 0$	$-3.9994e + 12$	0

$f(x) = e^x$	FD $_{h=10^{-3}}$	Err $_{h=10^{-3}}$	FD $_{h=10^{-4}}$	Err $_{h=10^{-4}}$	TP
$f(1)^{(1)} = e^1$	2.7182823	$4.5305e - 07$	2.7182818	$4.5306e - 09$	2.7182818
$f(1)^{(2)} = e^1$	2.7182821	$2.2654e - 07$	2.7182819	$3.7806e - 08$	2.7182818
$f(1)^{(3)} = e^1$	2.7182831	$1.3257e - 06$	2.7182701	$-1.1775e - 05$	2.7182818
$f(1)^{(4)} = e^1$	2.7182701	$-1.1775e - 05$	4.4408921	$1.7226e + 00$	2.7182818

# Differential Algebra — Radius of Convergence

---

The validity of the Taylor polynomial  $T_f^n$  representation for the real or complex *analytic function*  $f$  is characterized by the convergence of the remainder  $R_f^n$  in the neighborhood (open set) of the point  $a$ :

$$\lim_{n \rightarrow \infty} R_f^n(x; a) = \lim_{n \rightarrow \infty} f_a(x) - T_f^n(x; a) = 0$$

The *radius of convergence* of  $T_f^n(x; a)$  nearby the point  $a$  is given by:

$$\min_{\|x-a\|} \lim_{n \rightarrow \infty} R_f^n(x; a) \neq 0$$

By using the *mean value theorem* recursively we can derive the explicit Lagrange form of the remainder for some  $\xi$  strictly between  $x$  and  $a$ :

$$R_f^n(x; a) = \frac{f_a^{(n+1)}(\xi)}{(n+1)!} (x-a)^{n+1}$$

leading to the mean-value form of the *Taylor's theorem*:

$$f_a(x) = T_f^n(x; a) + R_f^n(x; a)$$

# Differential Algebra — Multivariate Polynomials

The generalization of Taylor polynomials to  $\nu$  variables  $X$  at order  $n$  nearby the point  $A$  in the  $\nu$ -dimensional domain of the function  $f$ , noted  $T_f^n(X; A)$ , has the following representation:

$$T_f^n(X; A) = \sum_{k=0}^n \frac{f_A^{(k)}}{k!} (X; A)^k = \underbrace{\sum_{k=0}^n \frac{1}{k!}}_{\text{orders}} \underbrace{\sum_{|\vec{m}|=k} \binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}} \Big|_A}_{\text{homogeneous polynomials}} (X; A)^{\vec{m}}$$

where the term  $\binom{k}{\vec{m}} = \frac{k!}{c_1! c_2! \dots c_\nu!}$  is the multinomial coefficient with  $\vec{m}$  the vector of  $\nu$  variables orders  $c_i, i = 1.. \nu$  in the monomial and  $|\vec{m}| = \sum_i c_i$  its total order. Again, we may mention that each term  $\frac{1}{k!} \binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}} \Big|_A$  corresponds strictly to a coefficient stored in the TPSA.

- ☞ **Generalized TPSAs** are TPSAs where  $\vec{m}_{\max}$  hold non-uniform orders.
- ☞ So far we have only been talking about **single standalone TPSA**.