Overview and Status of Linac Simulations with SelaV

Philipp Amstutz

FLASH2020+ S2E Workshop
2021-12-16

semi-Lagrangian Vlasov simulation

■ Vlasov codes solve the Vlasov equation, i.e. they find a solution for the phase space density (PSD) $\Psi(t,z): \mathbb{R} \times \mathbb{R}^{2n} \to \mathbb{R}$

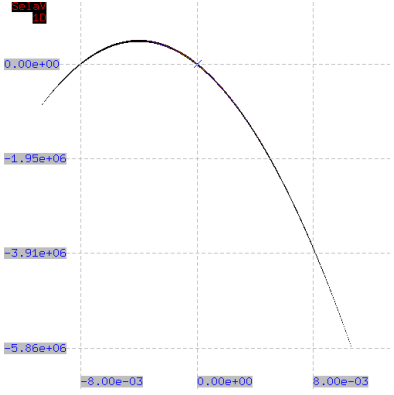
$$\partial_t \Psi - \{H[\Psi], \Psi\} = 0$$

■ semi-Lagrangian codes use the method of characteristics to construct the solution using the flow $\phi: \mathbb{R} \times \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ of the Hamiltonian $H: \mathbb{R}^{2n} \to \mathbb{R}$ (i.e. $\partial_t \phi = \{H[\Psi], \phi\}$)

$$\Psi(t_1, z) = \Psi(t_0, \phi_{t_0 \leftarrow t_1}(z))$$

■ implementation in a simulation code: store PSD values on a grid, update by back-tracking PS-coordinates and interpolate old PSD (a.k.a. Perron-Frobenius step)

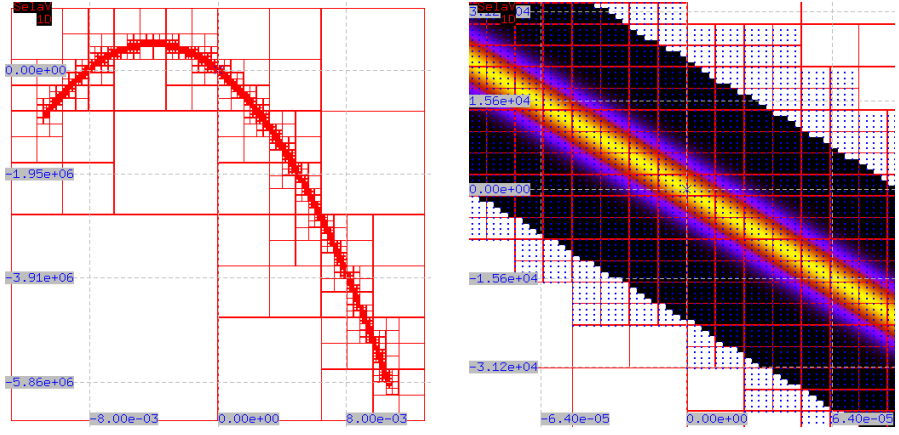
challenge: exotic densities in FELs



- longitudinal PSDs in FELs are exotic:
  - local energy variance is small compared to the energy range
  - curvature due to non-linearities from RF, collective effects
- "Ψ is exotic" $\iff$ supp(Ψ) is much smaller than its minimum bounding rectangle (MBR)
- problem for simulation codes: sampling Ψ on a homogeneous grid covering its MBR, includes huge empty areas

  $\implies$ too wasteful for high resolutions

solution: tree-based domain decomposition



- simulation domain is subdivided into a hierarchy of nested rectangles
- PSD values are only stored on grids (blue dots) covering the smallest rectangles (“leafs”)
  $\Longrightarrow$ minimizes sampling of $\overline{\text{supp}\Psi}$
- fundamental idea behind SelaV

the SelaV project

`libselav`

- C++ library
- implements arbitrary dimensional hypercube-tree data structure
  – dimension being a template parameter
- various interpolation routines (up to a C^2 third-order polynomial scheme)
- propagation routines implementing the Perron-Frobenius step
- collection of typical maps/flows
- physics-agnostic (except for some accelerator-physics specific maps/flows)

`selav1d`

- builds upon `libselav`
- implements a 1D vlasov code intended for (but not limited to) FEL linac simulations
- various PSD initialization routines
  – Gaussian, flat-top, analytic function, particle ensemble (e.g. Astra)
- flexible input language allows handling of PSDs, maps as objects

example: Vlasov-Poisson Equation / Landau damping

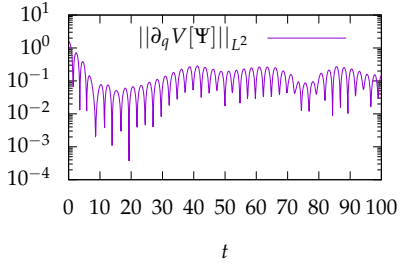
landau.inp

```
psi = psd_analytic(                          /* generate psd from analytic func. */
    "(1-0.5*cos(0.5*q))*exp(-0.5*p*p)/sqrt(2*pi)",
    limits=[-2*pi, -5, 2*pi,5],              /* simulation domain limits         */
    topology=1,                              /* C x R                            */
    depth=4,                                 /* depth of quad-tree               */
    nexp=5,                                  /* log2(sample points / dimension)  */
    interpolation=3                          /* 5th order C^2 interpolation      */
);

normalize(psi);                              /* psi <- psi / int_{R^2) psi dz    */

T=100;                                       /* simulation time                  */
DT=1e-1;                                     /* time step                        */

D = map_driftl(DT);                          /* define linear drift map          */

i=0; do(T/DT) {                              /* loop over time steps             */
    print(i);

    /* generate output */
    show(psi, file=format("%05g.ppm",i) );   /* write image of psi               */
    write_localmoments(psi,                  /* write energy-centroid, -variance */
        format("moments-%05g.dat",i),        /* -skewness, etc to file           */
        1
    );

    /* calculate maps */
    K = map_poisson1d(psi,factor=DT,         /* solve 1d poisson eq for psi      */
        file=format("field-%05g.dat",i)      /* also write solution to file      */
    );
    M = map_compose(K,D);                    /* M(z) = D(K(z))                   */

    /* propagate psi */
    psi = propagate(psi, M,                  /* psi <- psi( M^-1(z) )            */
        box="KEEP", center="KEEP"            /* keep simulation domain           */
    );

    i=i+1;                                   /* increment loop counter           */
};
```

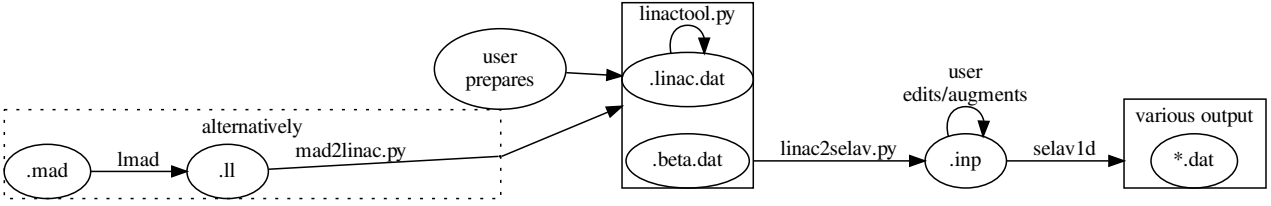
$$\partial_t \Psi - \{p^2/2 + V[\Psi], \Psi\} = 0$$

$$-\partial_q^2 V[\Psi] = \int_{\mathbb{R}} \Psi \mathrm{d}p - 1$$

movie landau



workflow for preparing a linac simulation



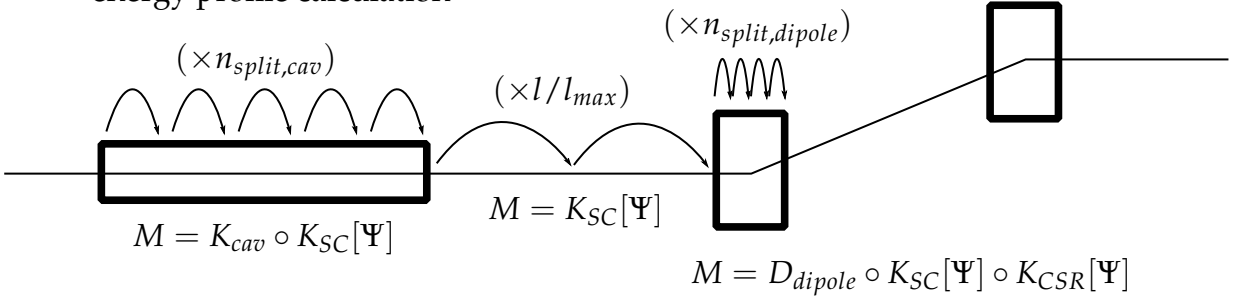
- basic idea: generate SelaV input file from simple linac description `.linac.dat` (plus beta function data `.beta.dat`)
- `.linac.dat` are ease to write manually or generate from existing data
  - syntax on next slide
  - optionally w/ some clean-up (remove/rename/merge elements) using `linactool.py`
- crucially, all lattice parameters (except basic geometry) are variables in the SelaV input file
  $\Longrightarrow$ available for scans
- resulting SelaV input file can then be edited/augmented as needed

linac2selav

■ linac decription syntax for `linac2selav.py`:

```
      cavity <name> <z0> <z1> <energy gain [eV]> <frequency [Hz]> <phase [rad]>
|     dipole <name> <z0> <z1> <strength [1/m] or rel. to 1st w/ same name>
| quadrupole <name> <z0> <z1> <strength [1/m^2]>
```

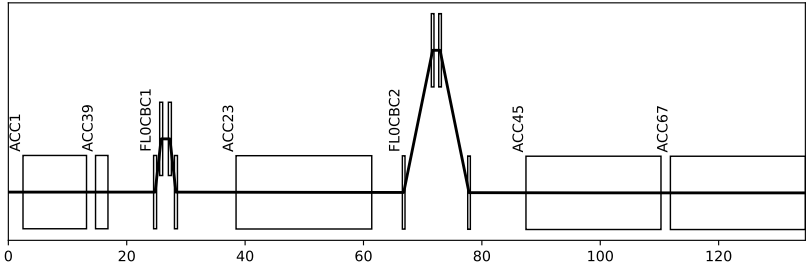
■ resulting SelaV input file takes care of:
- dispersion calculations
- spacecharge kicks in non-dispersive regions
- operator splitting (CSR and spacecharge) in dispersive regions
- energy profile calculation



example linac description: FLASH2020+ WP1

F2020P-WP1.linac.dat

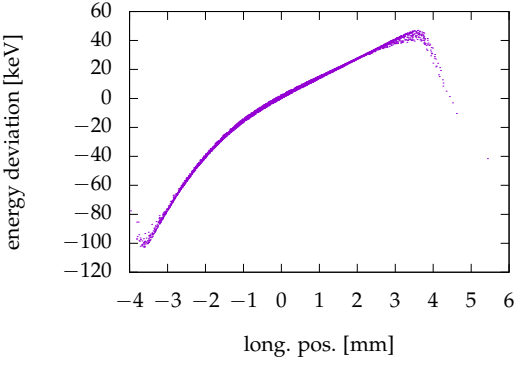
cavity	ACC1	2.4779	13.2078	159.624e6	1.3e9	0.1102
cavity	ACC39	14.7330	16.8310	-19.224e6	3.9e9	2.9431
dipole	FL0CBC1	24.5403	25.0468	0.5513		
dipole	FL0CBC1	25.5670	26.0736	-1		
dipole	FL0CBC1	27.0366	27.5431	-1		
dipole	FL0CBC1	28.0633	28.5698	+1		
cavity	ACC23	38.4707	61.3960	404.000e6	1.3e9	0.2276
dipole	FL0CBC2	66.5668	67.0137	0.1953		
dipole	FL0CBC2	71.4611	71.9080	-1		
dipole	FL0CBC2	72.7117	73.1586	-1		
dipole	FL0CBC2	77.6060	78.0529	+1		
cavity	ACC45	87.4489	110.2589	380.000e6	1.3e9	0.0
cavity	ACC67	111.8489	134.6494	420.000e6	1.3e9	0.0

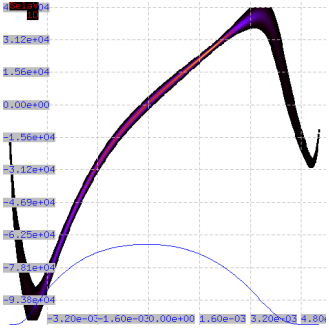


import/export particle ensembles

- densities can be imported from ensembles using the "local moments" method
  - divide particles in longitudinal slices around q_i
  - determine energy-centroids μ_i and -variances σ_i^2 of particles in slices
  - use interpolatants μ and σ to calculate

$$\Psi(q,p) = \frac{1}{\sqrt{2\pi\,\sigma(q)^2}} \exp\left(\frac{1}{2}\left(\frac{p-\mu(q)}{\sigma(q)}\right)^2\right)$$





- (N.B.: smooth kernel density estimation was not able to get rid of shot noise)
- export: efficient sampling via weighted tree descent

FLASH2020+ examples (WP1 w/ $\sigma_E = 10\mathbf{keV}$)

homogeneous

seeded

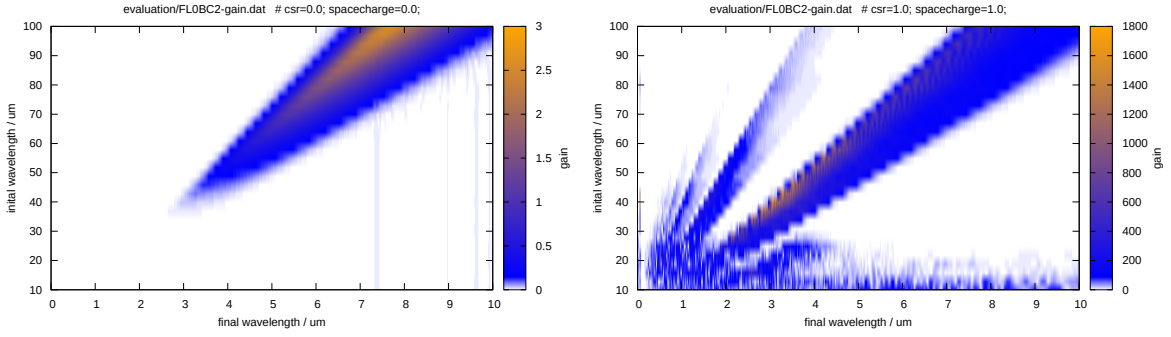
shot-noise

movie smooth

movie seed

movie noise

possible studies: 2D gain functions (old data)



- imprint inhomogeneity on the initially homogeneous PSD

$$\Psi^{(\mathrm{ini})}_{A,k_i}(q,p) = (1 + A\cos(k_i\, q))\Psi_0(q,p)$$

- run simulations for different k_i and determine 2D gain function

$$G_A(k_f,k_i) := \frac{\tilde{\rho}^{(\mathrm{fin})}_{A,k_i}(k_f)}{A}, \quad \text{with } \tilde{\rho} = \mathcal{F}\int_{\mathbb{R}} \Psi(\cdot,p)\mathrm{d}p$$

- linear gain function: $G(k) = \lim_{A\to 0} G_A(C\,k,k)$

Thanks!