Intensity effects in the longitudinal plane

A. Lasheen, D. Quartullo

23rd HEADTAIL Development meeting – 25/07/14

Outline

- Slicing (beams.slices)
  - Slicing methods
  - Extra methods
- Impedance (impedances.longitudinal_impedance)
  - Coordinates
  - General methods
  - Sources
  - Induced voltage calculation
- Examples
- Conclusion
- Forthcoming developments

Slicing

```
# DEFINE SLICES--------------------------------------------------------------

number_slices = 100
slice_beam = Slices(number_slices, cut_left = - 5.72984173562e-07 / 2,
                    cut_right = 5.72984173562e-07 / 2, coord =
                    "tau", mode = 'const_space_hist')
```

Slicing

Slicing – Slicing methods

slice_constant_space	slice_constant_space_histogram	slice_constant_charge
- Profile obtained through the method developed by Hannes and Kevin before, by sorting the particles and counting the particles - Needed in order to use the compute_statistics method in slice module	- Profile obtained from the numpy histogram function - Only returns the number of particles per slice with a constant bin size for a constant frame size	- Bin size adapted to have the same number of particles in each bin - Difficult to compute induced voltage in frequency domain with this slicing…

Only one used so far for longitudinal effects

Slicing – Extra methods

- Slicing can be done in θ, z and t (***t* is preferable** in order to calculate induced voltage)

- Other methods are included in order to calculate:

  - Beam spectrum

  - Beam profile derivative

  - Statistics

Impedance

```
# LOAD IMPEDANCE TABLES-------------------------------------------------------

var = str(kin_beam_energy / 1e9)

# ejection kicker
Ekicker = np.loadtxt('ps_booster_impedances/ejection kicker/Ekicker_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

Ekicker_table = Longitudinal_table(Ekicker[:,0].real, Ekicker[:,1].real, Ekicker[:,1].imag)

# ejection kicker cables
Ekicker_cables = np.loadtxt('ps_booster_impedances/ejection kicker cables/Ekicker_cables_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

Ekicker_cables_table = Longitudinal_table(Ekicker_cables[:,0].real, Ekicker_cables[:,1].real, Ekicker_cables[:,1].imag)

# KSW kickers
KSW = np.loadtxt('ps_booster_impedances/KSW/KSW_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

KSW_table = Longitudinal_table(KSW[:,0].real, KSW[:,1].real, KSW[:,1].imag)

# resistive wall
RW = np.loadtxt('ps_booster_impedances/resistive wall/RW_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

RW_table = Longitudinal_table(RW[:,0].real, RW[:,1].real, RW[:,1].imag)

# indirect space charge
ISC = np.loadtxt('ps_booster_impedances/Indirect space charge/ISC_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

ISC_table = Longitudinal_table(ISC[:,0].real, ISC[:,1].real, ISC[:,1].imag)

# steps
steps = Longitudinal_inductive_impedance(34.6669349520904 / 10e9) # input in [Ohm/Hz
```

Impedance sources

Examples – PS Booster

```
# INDUCED VOLTAGE FROM IMPEDANCE---------------------------------------------

# impedance to be used for ind_volt calculation through the profile spectrum
sum_impedance = [Ekicker_table] + [Ekicker_cables_table] + [KSW_table] \
                + [RW_table] + [F_C_table] + [ISC_table]

# impedance to be used for ind_volt calculation through the profile derivative
sum_slopes_from_induc_imp = (376.730313462 * general_params.T0[0, 0]) / \
        (my_beam.beta_r * my_beam.gamma_r**2) - \
        34.6669349520904 / 10e9    # direct space charge plus steps, in [Ohm/Hz]

ind_volt_from_imp = Induced_voltage_from_impedance(slice_beam, "on", sum_impedance, 2e5,
                sum_slopes_from_induc_imp, mode = 'spectrum + derivative')

# MAP---------------------------------------------------------------------

map_ = [slice_beam] + [ind_volt_from_imp] + [ring_RF_section]
```

Impedance sum

Induced voltage

&

Track

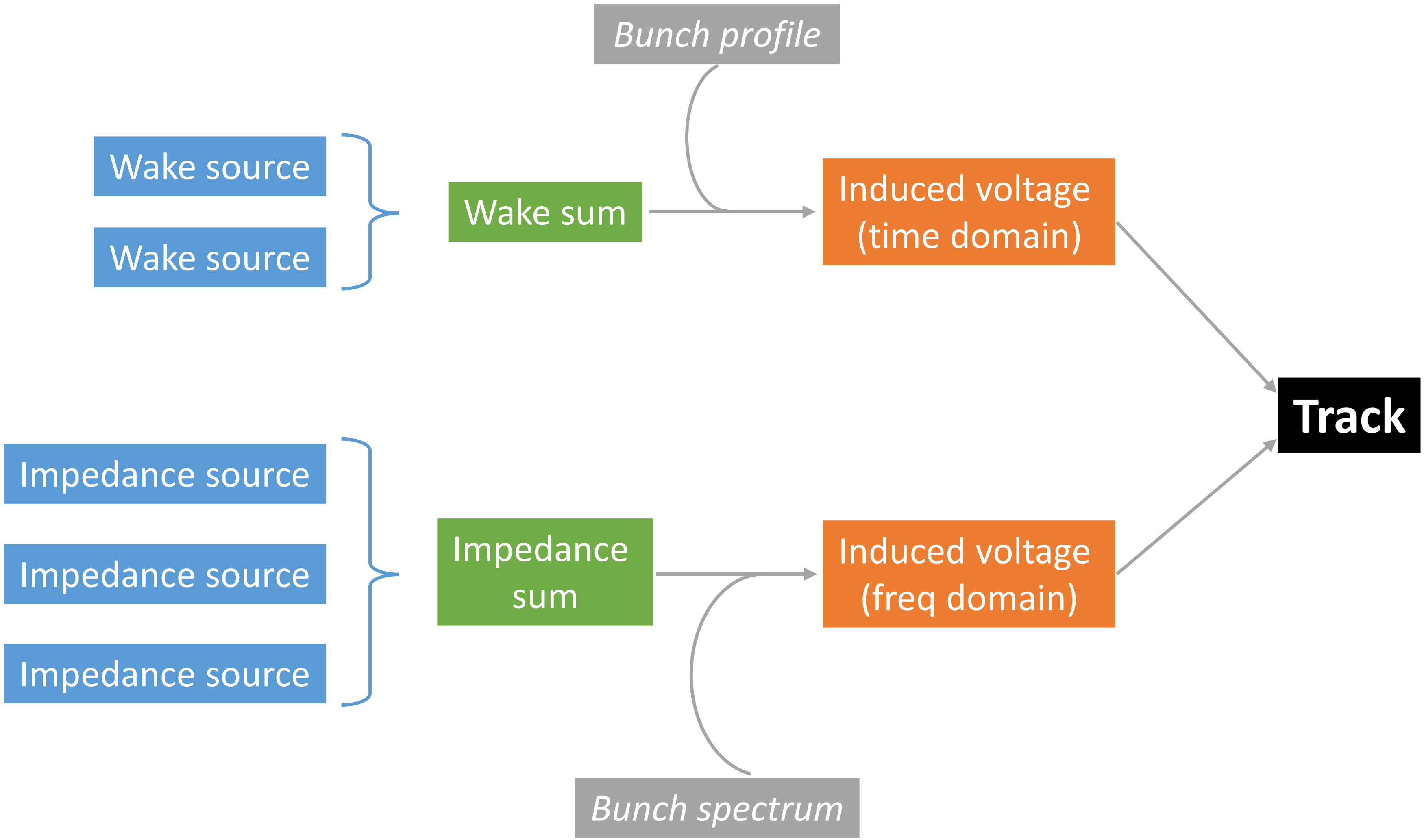
Impedance - Coordinates

- The coordinates used are:
  - Time for wakes
  - Frequency for impedances

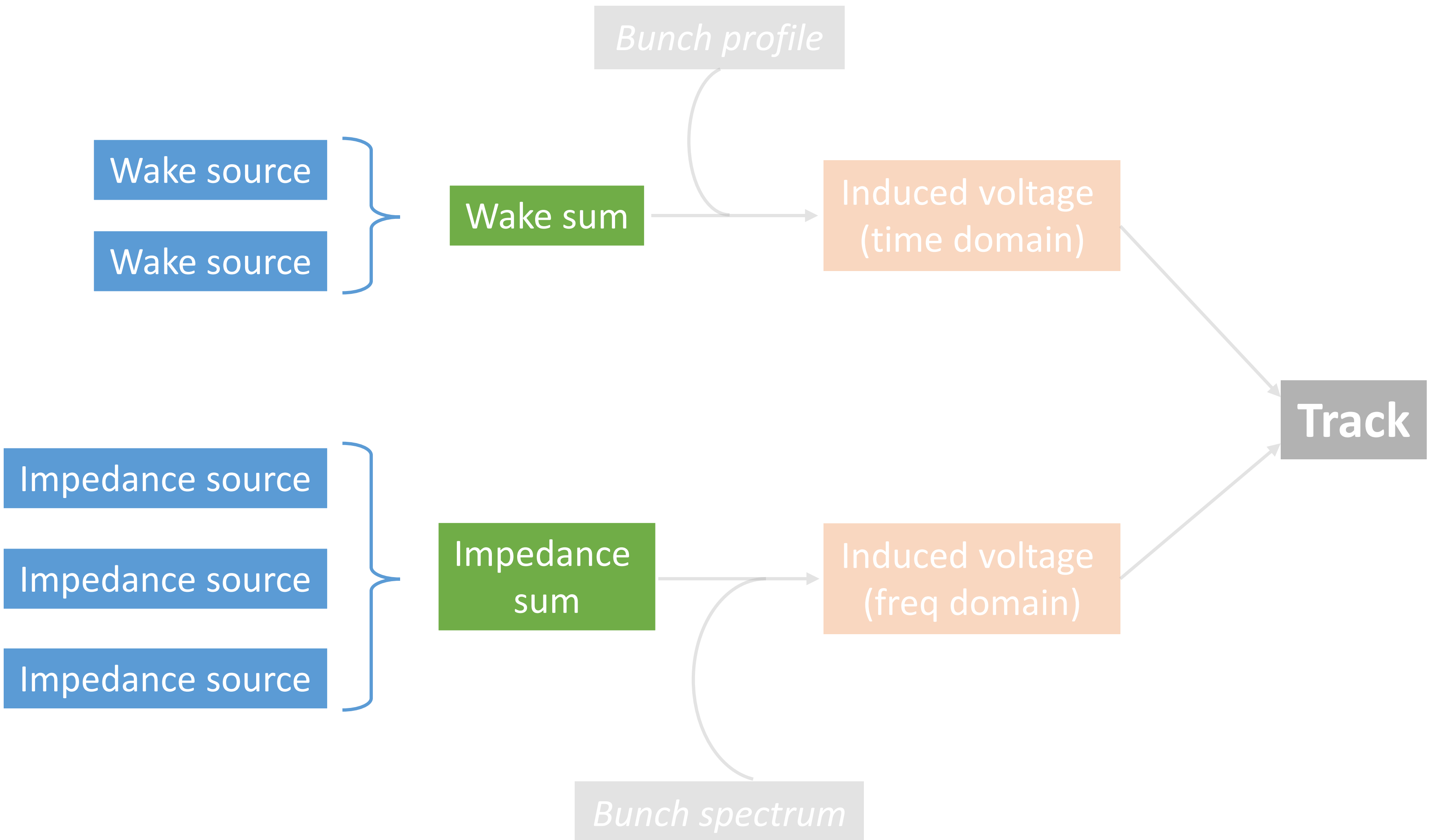
- Wakes are expressed in t rather than in z in order to be independent of β in case of acceleration. This allows to pre-process the wakes and gain time during tracking.

- One can still use induce voltage calculation in other coordinates, implying a recalculation of the wake every turn.

Impedance – General method



Impedance – General method



Preprocessed if slicing and wakes are calculated in time, and impedance in frequency

Impedance – Sources

- ❑ Resonators (wakes and impedances)

- ❑ Travelling wave cavities (wakes and impedances)

  - ➢ *Warning*: high Q resonators and cavity models require a very small frequency resolution

- ❑ Tables

- ❑ Constant imaginary Z/n
  - ❑ Two methods are possible, to calculate: as any other impedance with inverse FFT, or via the derivative of the line density

  - ➢ Warning: Noise might be an issue to smooth, but all cases of simulations does not allow a simple smoothing…

➢ Different sources can be summed for faster tracking if they have the same resolution, or used one by one.

Impedance – Induced voltage

- Time domain calculation can be done with two methods:
  - **Matrix convolution** (method by Hannes) that works for the constant_space and constant_charge slicing methods
  - The **numpy convolve function** that works for constant_space and constant_space_histogram slicing
- Frequency domain calculation
  - Inverse fft of the bunch spectrum times the impedance
  - For the constant imaginary Z/n, the beam profile derivative is multiplied by the impedance

Conclusion

- A lot of options are possible:
  - Diverse coordinates
  - Diverse options for the slicing
  - Diverse options for the impedance sources
  - Diverse methods to calculate the induced voltage and track

- Some are faster/more complete than the other:
  - The time/frequency coordinates are good for preprocessing of the wakes/impedance
  - As a rule of thumb, numpy histogram and convolve functions are faster to calculate
  - The constant_space and constant_charge slicing are useful in the case where you need to compute statistics for each slice (via the sorting)

Examples – PS Booster

```
# DEFINE SLICES-----------------------------------------------------------------

number_slices = 100
slice_beam = Slices(number_slices, cut_left = - 5.72984173562e-07 / 2,
                    cut_right = 5.72984173562e-07 / 2, coord =
                    "tau", mode = 'const_space_hist')
# LOAD IMPEDANCE TABLES---------------------------------------------------------

var = str(kin_beam_energy / 1e9)

# ejection kicker
Ekicker = np.loadtxt('ps_booster_impedances/ejection kicker/Ekicker_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

Ekicker_table = Longitudinal_table(Ekicker[:,0].real, Ekicker[:,1].real, Ekicker[:,1].imag)

# ejection kicker cables
Ekicker_cables = np.loadtxt('ps_booster_impedances/ejection kicker cables/Ekicker_cables_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

Ekicker_cables_table = Longitudinal_table(Ekicker_cables[:,0].real, Ekicker_cables[:,1].real, Ekicker_cables[:,1].imag)

# KSW kickers
KSW = np.loadtxt('ps_booster_impedances/KSW/KSW_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

KSW_table = Longitudinal_table(KSW[:,0].real, KSW[:,1].real, KSW[:,1].imag)

# resistive wall
RW = np.loadtxt('ps_booster_impedances/resistive wall/RW_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

RW_table = Longitudinal_table(RW[:,0].real, RW[:,1].real, RW[:,1].imag)

# indirect space charge
ISC = np.loadtxt('ps_booster_impedances/Indirect space charge/ISC_' + var + 'GeV.txt'
        , skiprows = 1, dtype=complex, converters = dict(zip((0, 1), (lambda s:
        complex(s.replace('i', 'j')), lambda s: complex(s.replace('i', 'j'))))))

ISC_table = Longitudinal_table(ISC[:,0].real, ISC[:,1].real, ISC[:,1].imag)

# steps
steps = Longitudinal_inductive_impedance(34.6669349520904 / 10e9) # input in [Ohm/Hz
```

Slicing

Impedance sources

Examples – PS Booster

```
# INDUCED VOLTAGE FROM IMPEDANCE---------------------------------------------

# impedance to be used for ind_volt calculation through the profile spectrum
sum_impedance = [Ekicker_table] + [Ekicker_cables_table] + [KSW_table] \
                + [RW_table] + [F_C_table] + [ISC_table]

# impedance to be used for ind_volt calculation through the profile derivative
sum_slopes_from_induc_imp = (376.730313462 * general_params.T0[0, 0]) / \
        (my_beam.beta_r * my_beam.gamma_r**2) - \
        34.6669349520904 / 10e9    # direct space charge plus steps, in [Ohm/Hz]

ind_volt_from_imp = Induced_voltage_from_impedance(slice_beam, "on", sum_impedance, 2e5,
                sum_slopes_from_induc_imp, mode = 'spectrum + derivative')

# MAP----------------------------------------------------------------------

map_ = [slice_beam] + [ind_volt_from_imp] + [ring_RF_section]
```

Impedance sum

Induced voltage

&

Track

Examples – Files in GitHub

- Link to GitHub

- Examples in files:
  - _Wake_impedance.py
  - _Impedance_ps_booster.py

Forthcoming developments

- ❑ Working on the correlation between the beam and slice classes
- ❑ Cleaning and optimizing
- ❑ Documentation
- ❑ Testing M. Migliorati’s method (MuSiC code)
- ❑ Bunch generation matched with intensity effects
- ❑ Smoothing