MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

Update on first flight neutral model implementation in EMC3-Lite

email: robert.davies@ipp.mpg.de

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

This research was supported in part by the Helmholtz International Laboratory for Optimized Advanced Divertors in Stellarators (HILOADS).

Motivation

- **Divertor performance** is tied to **neutral density** in the divertor/sub-divertor region, and **geometric closure** plays an important role
- EMC3-Eirene models neutrals but relatively expensive. Might be able to understand/optimise for geometric closure using simpler models
- Easiest approximation: Neutrals travel ballistically (long mean-free path limit).
  - **Relatively cheap (neutral path independent of plasma density, temperature)**
  - **Possibly valid in W7-X cases (open divertor)**

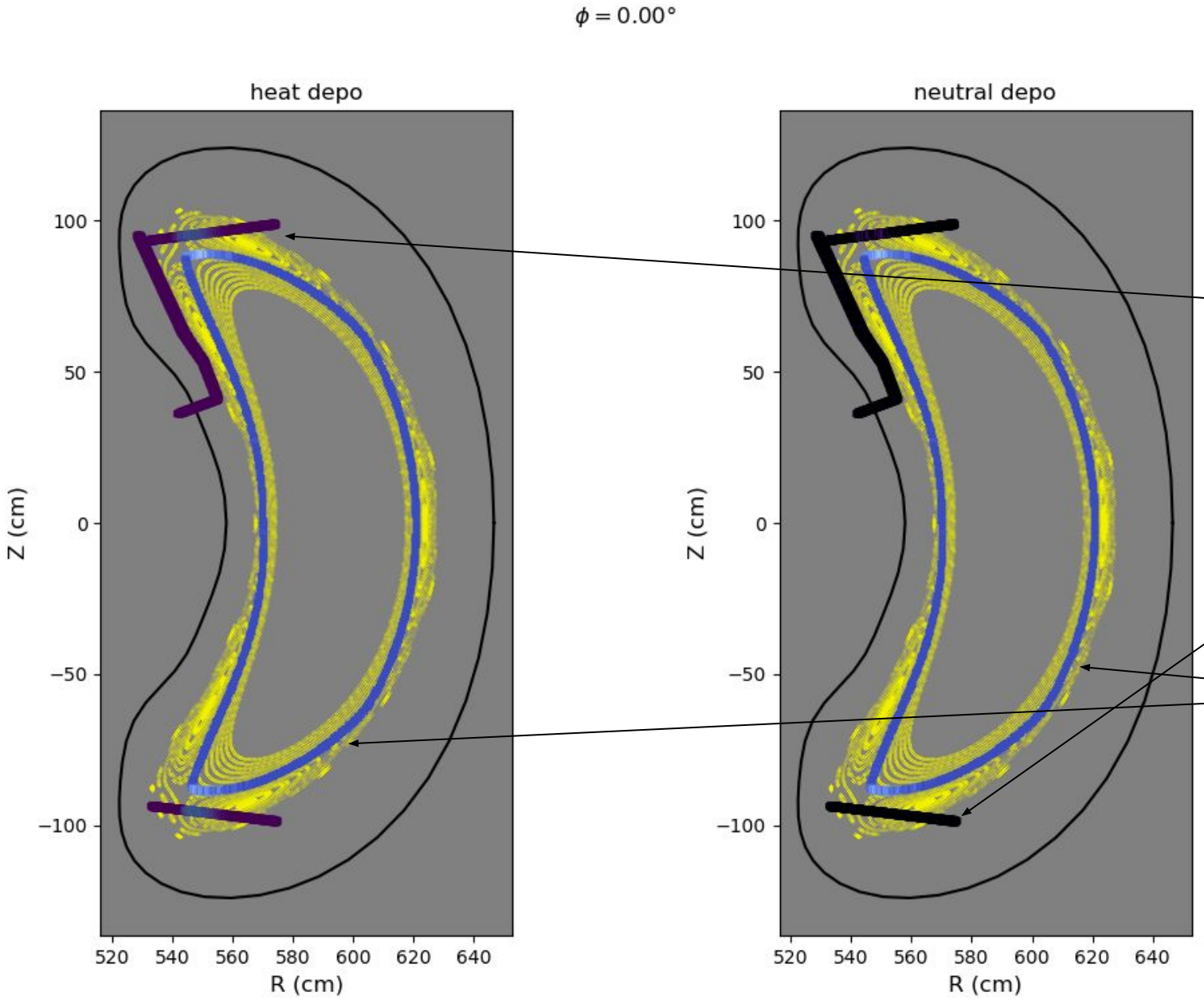
The physical model implemented in EMC3-Lite

- Neutrals are born on PFCs only (surface recombination)
- Neutral source on PFC proportional to heat load
  - **Probably reasonable in attached cases**
- Neutrals travel like rays/ballistically (long mean free path limit)
- Neutral velocity distribution obeys Lambert's cosine rule
- Neutrals travel until they hit another PFC, or the LCFS, or the end of the half field period domain

A few implementation details

- **Inputs**: the output of a deposition calculation
- **Output**: neutral deposition information after 1 flight
- Related code developments:
  - **Python scripts for postprocessing**
  - **Scheme for working out load on front vs back of PFC cells**
    - **Probably solves the “bad target connection length” issue**
    - **Contact Bob if you have a “bad target connection length” example to test the fix**
  - **A small testing framework**
    - **Allows developers to write and run tests to verify the code**

Preliminary results

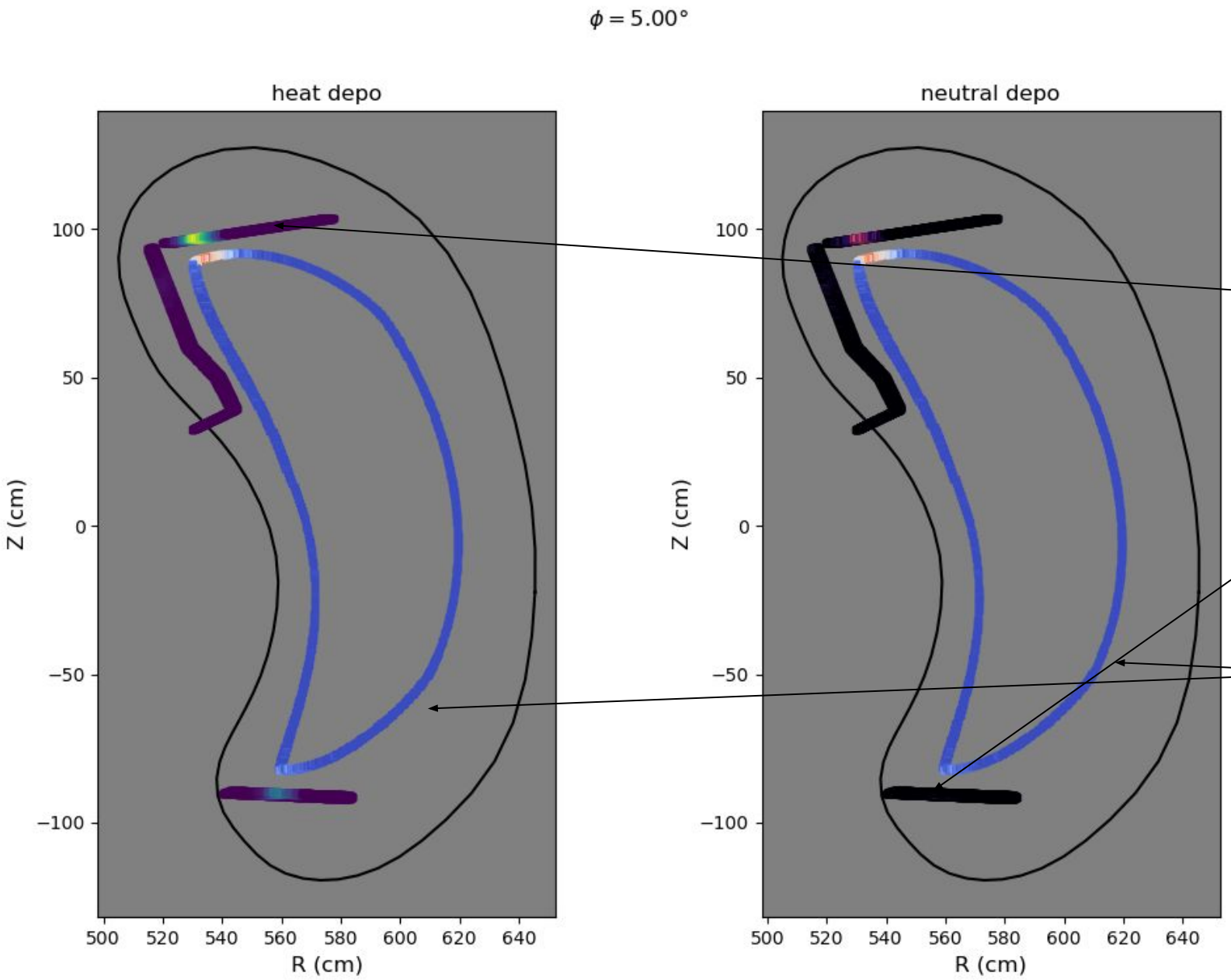


PFCs, heat deposition
purple=low power/unit area
green/yellow= high power/unit area

PFCs, neutral deposition
black=low neutrals/unit area
orange = high neutrals/unit area

LCFS, neutral deposition
blue=low neutrals/unit area
red= high neturals/unit area

Preliminary results

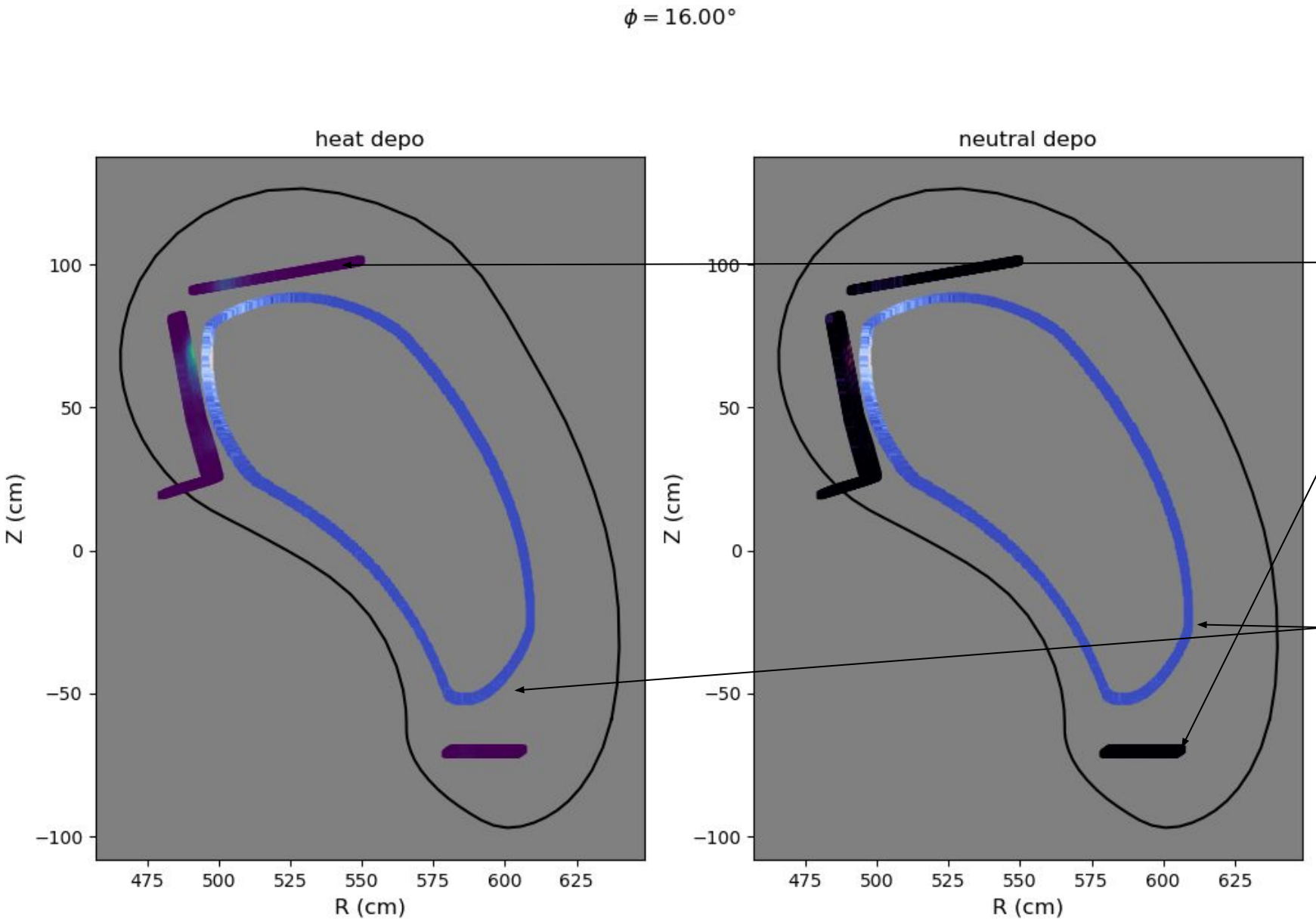


PFCs, heat deposition
purple=low power/unit area
green/yellow= high power/unit area

PFCs, neutral deposition
black=low neutrals/unit area
orange = high neutrals/unit area

LCFS, neutral deposition
blue=low neutrals/unit area
red= high neturals/unit area

Preliminary results



PFCs, heat deposition
purple=low power/unit area
green/yellow= high power/unit area

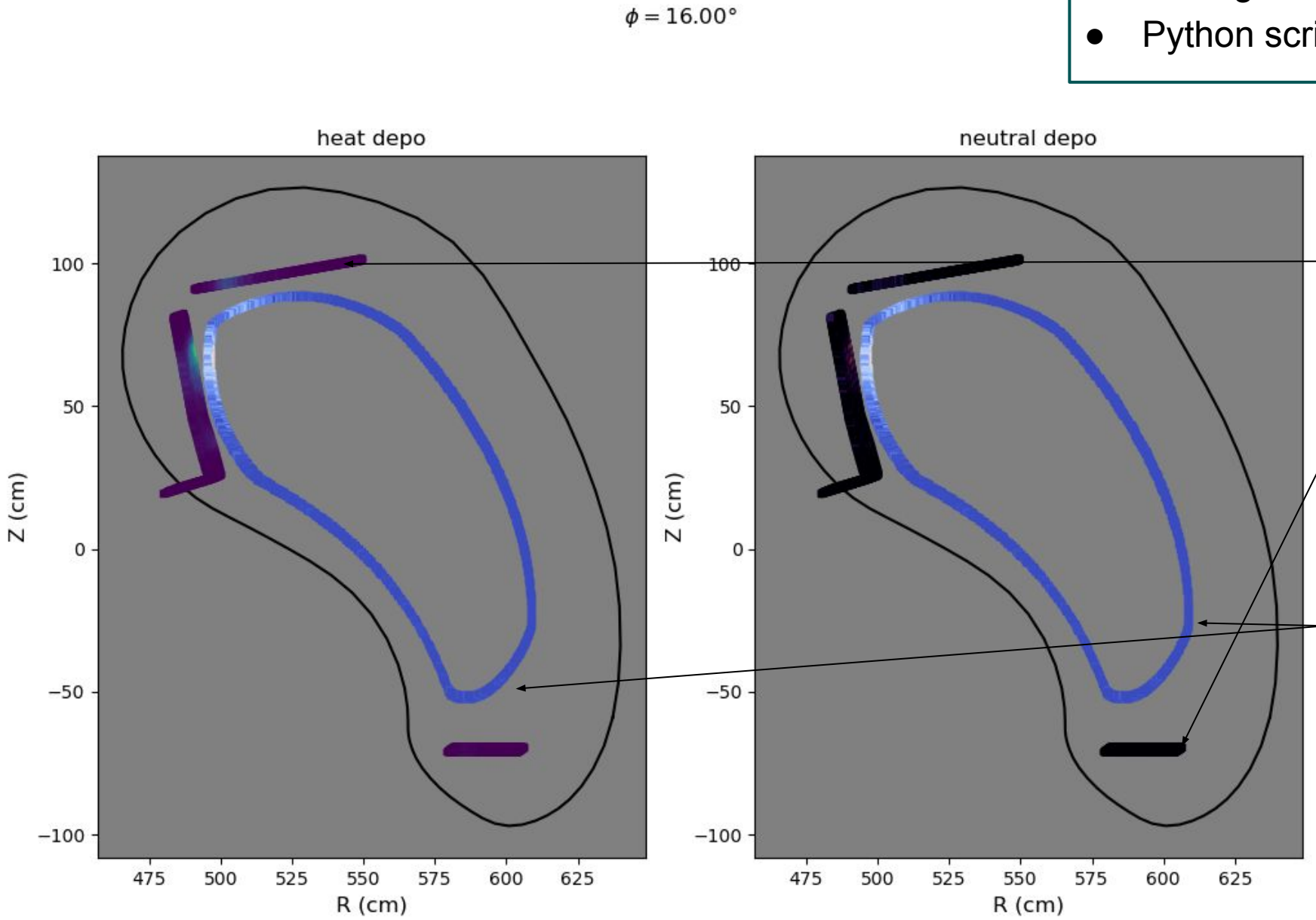
PFCs, neutral deposition
black=low neutrals/unit area
orange = high neutrals/unit area

LCFS, neutral deposition
blue=low neutrals/unit area
red= high neturals/unit area

Preliminary results

Ongoing work

- Writing & running tests
- Python scripts for postprocessing



PFCs, heat deposition
purple=low power/unit area
green/yellow= high power/unit area

PFCs, neutral deposition
black=low neutrals/unit area
orange = high neutrals/unit area

LCFS, neutral deposition
blue=low neutrals/unit area
red= high neturals/unit area

APPENDIX

A little test framework in EMC3-Lite

- **Motivation**: It's relatively easy to introduce bugs in a code, and it's nice to catch them early on. Code tests helps verify the code is doing what we expect, and makes debugging easier.
- While implementing the neutral FF model, I added a script which runs tests on the new subroutines and functions. Some of these tests work entirely in fortran. Others write text output, which are then analysed by python scripts
- NB this is a quick and dirty implementation of a testing framework. More sophisticated options exist (e.g. the "pfunit" toolkit), but require more time to implement/maintain
- Bob's plan: implement tests as and when needed. The ideal case is to have the code fully covered by tests, but I'm unlikely to achieve this alone.
- ***EMC3-Lite users/developers***: Please **feel free** to add your own tests to the branch. Email Bob with questions etc.

Some implementation details

Inputs and outputs

- Inputs:
  - **The heat deposition on the PFCs (“TARGET_PROFILES”)**
  - **Information on the side on which heat is falling (“TARGETS_SIDE1_FRACTION”)**
  - **The LCFS (from the GRID_3D.txt file)**
- Output: the neutral deposition on the PFCs (“TARGET_PROFILES_NEUTRALS”)

Quadrilateral vs triangular cells

- EMC3-Lite normally specifies each PFC as a logically rectangular grid of quadrilateral cells (the geometry specified in the “Kisslinger” geometry format, which can be upsampled by the user). Each quadrilateral is specified by 4 points. In general, these cells are **non-planar** (the cells are curved, because 4 points, in general, do not lie in a plane).
- Non-planar cells make the geometry of the problem rather complicated (finding normal vectors, intersection tests etc.) To simplify things, the neutral implementation splits each quadrilateral cell into 2 triangular cells. The triangular cells are planar.
- ***Advice to users***: ensure that the TARGET_PROFILES data is sufficiently upsampled (largest errors for the largest and most non-planar cells)

Initialising the Monte Carlo neutral particles

- Use random number generator to assign each Monte Carlo neutral’s inital position to a triangle. The probability of assignment for each triangle is proportional to its heat load (NB power on the target, not power per unit area). NB the number of triangular cells can easily exceed the number of MC neutrals, with most cells having a source of 0 MC neutrals.
- Currently the position is set to be the centre of the triangle. Future work: make initial position random on the triangle.
- The velocity unit vector (i.e. direction of travel of the neutral) is set using random numbers, which a probabilistic distribution in accordance with Lambert’s cosine rule.

Some implementation details

Checking the neutral/target interaction

- For each neutral, the code finds the first “target” intersected by the neutral. This target can be:
  - **A triangular PFC cell**
  - **The LCFS (which is discretised as a bunch of triangular cells)**
  - **The end of the half field period domain**
- Currently, the code loops over all targets to check for (1) and intersection (2) the distance travelled by the neutral to the target. The neutral hits the closest intersecting target
  - **NB this is relatively slow. Could speedup using some kind of grid scheme**
  - ***Advice to developers*: Address this first, if speed is a problem**
- MC neutral is absorbed by PFC cells and LCFS. At the ends of the half field period domain, boundary condition is applied and the particle is re-emitted from where it leaves

Direction of the normal vector/front-back cells

- Old behaviour: PFC cells do not know on which “side” they are receiving heat flux
- New behaviour: Information on which side the PFC cell receives heat flux is stored in the depo calculation and written to a file (“TARGETS_SIDE1_FRACTION”)
- The neutral deposition module uses this information to set the source correctly

The implementation - pseudocode

```
Subroutine NEUTRAL_DEPOSITION_MC()


 call INIT_NEUTRAL()
 call SET_MC_PARTICLES_PER_TRIANGLE()

 ! Loop over each triangular PFC cell
 do I_TRIANGLE = 1, ALL_TRIANGLES
       if (# MC particles for this triangular > 0) then
             ! Loop over each Monte Carlo neutral for this cell
             do I_NEUTRAL = 1, # MC particles
                   call make_and_run_neutral_particle(I_TRIANGLE)
             end do
       end if
  end do

  ! Write output to file and diagnostic information to terminal
  call WRITE_NEUTRAL_OUTPUT()

  ! Close the grids and deallocate memory
  call TERMINATE_NEUTRAL()

END Subroutine NEUTRAL_DEPOSITION_MC
```

```
Subroutine INIT_NEUTRAL()
<Get the input.ctr data>
call get_pfc_data ! Get PFC geometry and heat deposition
call init_lfcs_triangles ! Get the LCFS and discretise into triangles
call init_hfp_domain_ends !  Create the toroidal ends of the domain
```

```
Subroutine SET_MC_PARTICLES_PER_TRIANGLE()
! Use the information on the heat loads deposition and the total number
of neutral Monte Carlo particles to be simulated to get the number of
Monte Carlo particles emitted by each triangular cell.

! Does this by generating a random number for each MC particle and
then binning into a cell. Each triangular cell has a bin, and the bin width
is proportional to the heat deposition on the cell.
```

```
Subroutine make_and_run_neutral_particle(I_TRIANGLE)
! Create a neutral MC particle born by triangular cell I_TRIANGLE.
! Does this by selecting a point on the triangular cell as the initial point,
! and generating a velocity vector (set using random number generator,
! statistically obeying Lambert's cosine rule).
call init_neutral_particle
! follow its trajectory and record its deposition (where it hits). what the
! Currently does this by looping over all other triangular cells (PFC cells
! + LCFS cells + toroidal domain ends). Could be made faster.
call run_neutral_particle
```

A little test framework in EMC3-Lite - pseudocode

test program

```
Program EMC3_LITE_TESTS

  call TEST_TARGET_PROFILES_OPEN()
  call test_triangles_out()
  call test_lambert_angles()
  call test_velocity_unit_vector()
  call test_intersection()
  call test_neutrals_integrated_tests()

END Program EMC3_LITE_TESTS
```

test subroutines

```
! Tests the subroutine TARGET_PROFILES_OPEN
subroutine TEST_TARGET_PROFILES_OPEN

  ! Open a TARGET_PROFILES file which I made earlier.
  call TARGET_PROFILES_OPEN('TARGET_PROFILES')

  !! Now write the data to file. If TARGET_PROFILES_OPEN
  !! is correct, then the output file TARGET_PROFILES_TEST_OUT
  !! should be exactly the same as the input file TARGET_PROFILES
  call TARGET_OUT('TARGET_PROFILES_TEST_OUT')
  call TARGET_CLOSE()
end subroutine TEST_TARGET_PROFILES_OPEN
```

```
subroutine test_triangles_out
  ! Open a TARGET_PROFILES file which I made earlier.
  call TARGET_PROFILES_OPEN('TARGET_PROFILES')

  ! Turn the quadrilateral cells into triangular cells
  call INIT_TRIANGLES()

  ! Force the neutral deposition to be equal to the heat deposition
  NEUTRAL_E_TRI = DEPO_E_TRI

! Write triangular depo data to file. If init_triangles and triangles_out
are behaving correctly, then the heat deposition on the quadrilateral
cells in TARGET_PROFILES should match the deposition on the
triangular cells in TARGET_PROFILES_TRI_TEST_OUT
 call TRIANGLES_OUT('TARGET_PROFILES_TRI_TEST_OUT')
 call TARGET_CLOSE()
 call TERMINATE_TRIANGLES()
end subroutine test_triangles_out
```

python scripts for postprocessing

```
def test_targetprofiles_match()

quad_cell_data = open("TARGET_PROFILES")
tri_cell_data = open("TARGET_PROFILES_TRI_TEST_OUT")

<Check # PFCs and # cells are the expected values>
<Check the total deposition on each PFC within tolerance>
for pfc in all_pfcs:
        <Compare PFC metadata>
        for quadtileral_cell in pfc:
                <Compare heat depo with depo on the 2
                 corresponding triangular cells>
```