PSA-Diff-NearSlack-Dev

February 4, 2026

1 Numerical calculation of PSA levels

1.1 Table of contents

1.2 Notes

Nothing at the moment.

Return to ToC: »

1.3 Introduction

Model a prostate tumour and the resulting production of prostate specific antigen (PSA). Define a "body", a rectangular 3D grid of "cells". Make some of them prostate, some tumour, and some blood cells. Surround these with a "border", an outer layer that could be used to simplify the application of boundary conditions during calculation of the changes in PSA concentration. The overall size of the body does not change, so make this large enough to encompasss some tumour growth! PSA is produced at a low rate in the prostate and at a higher rate in the tumour cells and diffuses through these cells into the blood. The PSA also decays and the rate of decay is higher in the blood than in the prostate and tumour cells.

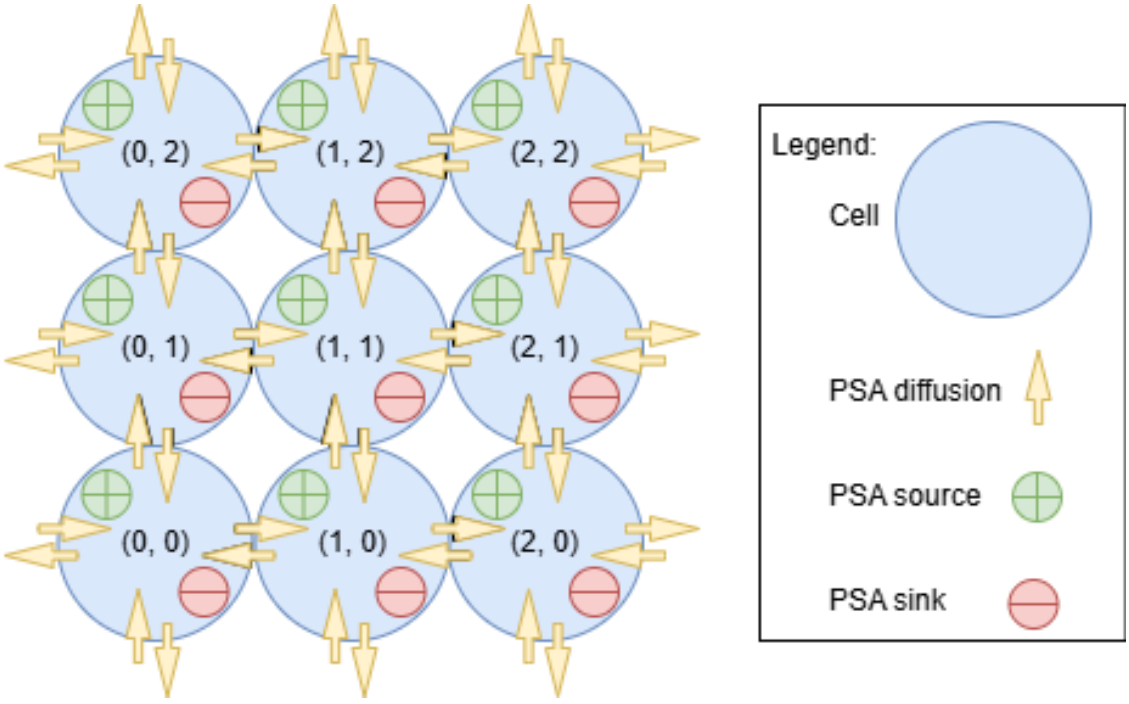


Figure 1 *Illustration in two dimensions of rectangular grid of cells, with flow of PSA from cell to cell and PSA generation and decay.*

Return to ToC: »

Model diffusion as transfer of PSA from one cell (1, 1) to its neighbours (0, 1), (1, 0), (2, 1) and (1, 2) in a time step of length δt. The amount transferred is proportional to the PSA concentration in the cell. the length of the common boundary and of the time step. Similarly, the neighbouring cells transfer PSA into the central cell. Labelling as ρ_{10} the PSA level in cell (1, 0), as ρ_{11} the density in cell (1, 1) and so on, and assuming all the boundaries are of the same length, the change in the quantity of PSA in the central cell in the time interval δt can then be expressed as:

$$\delta_{11} = \delta t \, \beta \, (\rho_{01} + \rho_{10} + \rho_{21} + \rho_{12} - 4\rho_{11}) \, .$$

Here, β is a constant of proportionality (which incorporates factors like the length of the cell boundary, its permeability etc.).

Allow for the generation of PSA in the cells; σ_{11} is the amount of PSA produced in cell (1, 1) per unit time:

$$\delta_{11} = \delta t\, \beta\, (\rho_{01} + \rho_{10} + \rho_{21} + \rho_{12} - 4\rho_{11}) + \delta t\, \sigma_{11}.$$

The rate of generation of PSA is different in different cell types.

A further necessary component is PSA decay. Adding this, the above equation becomes:

$$\delta_{11} = \delta t\, \beta\, (\rho_{01} + \rho_{10} + \rho_{21} + \rho_{12} - 4\rho_{11}) + \delta t\, \sigma_{11} - \frac{\delta t}{\tau}\rho_{11}$$

Here, τ is the lifetime of PSA. It is different in tumour or prostate cells and in the blood stream.

The amount of PSA in each of the neighbouring cells changes in the same manner.

This model can be extended to three dimensions. Using an obvious extension of the notation (and assuming the areas connecting the cells are the same), we have:

$$\delta_{111} = \delta t\, \beta\, (\rho_{011} + \rho_{101} + \rho_{211} + \rho_{121} + +\rho_{110} + \rho_{112} - 6\rho_{111}) + \delta t\, \sigma_{111} - \frac{\delta t}{\tau}\rho_{111}.$$

As the blood flows past the prostate and tumour, the PSA diffusing into the blood is shared between a large number of cells, resulting in a dilution factor for blood cells. Further, the (effective) diffusion between blood cells is not the same as in "static" cells. Assume that the diffusion in blood cells is essentially instantaneous. (Model this by defining a layer of blood cells adjacent to the prostate/tumour, diffuse PSA into these in the standard way, find the average PSA level in this layer, and assign it to all blood cells.)

The number of cells in the body may change. In particular, the tumour may grow. Assume for now that cell division happens at the surface of the tumour. The new cell is created next to a tumour cell and the displaced cell moves to another adjacent "neighbour" position. The neighbour moves to the nearest site outside the tumour and prostate. This last cell will extend into the blood so the overall prostate/tumour size will increase.

Cell death has yet to be implemented, but cells, in particular tumour cells, do die. Consider whether to shrink the prostate/tumour when this happens, or to leave a (blood-filled?) void.

Return to ToC: »

1.3.1 Cell type codes

Cell types are coded as follows.

In `body`:

Description	Code	Notes
Unspecifed	0	
Test	-1	
Border	1	Border of body, thickness 1 cell
Blood	2	
Prostate	3	
Tumour	4	

In `body_steer`:

Description	Code	Notes
Unspecified	0	
Surface	1	Layer of cells immediately outside tumour
Duplicate	2	Cell to be replicated
Exterior	3	Blood layer outside prostate/tumour
End	4	Position to which cell overwritten by duplicate is moved
Neighbour	5	Position to which duplicate written

Return to ToC: »

1.3.2 Increase tumour size - making duplications

Find the "surface" cells that lie around the tumour (i.e. cells for which one above is not tumour and/or one below is not tumour and/or one to the left is not tumour and/or one to the right is not tumour).
Choose one surface cell to be the position into which the replicated/duplicated cell goes.
Find the "end" cell, the blood cell closest to the cell to be duplicated.
The codes for these cells are stored in the `body_steer` array.
Find a "neighbour" cell, adjacent to the cell that will accept the duplicate (change cell code in `body_steer` appropriately).
Copy the cell type originally in the duplicate position into the neighbour position and the original neighbour into the end cell position (in `body`).
Create a new tumour cell (in `body`) at the duplicate position.

Return to ToC: »

1.4 Load libraries and initialise random number generator

Return to ToC: »

[1]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#
import matplotlib.cm as cm
rng = np.random.default_rng()
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:34.797460

Date and time 2026-02-04 10:29:35.370185
Time since last check is 0:00:00.572725
```

1.5 Functions

1.5.1 List body data

Return to ToC: »

[2]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def list_body_sums():
    '''
    List numbers of different type of cells in body and body_steer arrays
    '''
    n_blood = np.sum(body == c_blood)
    n_prostate = np.sum(body == c_prostate)
    n_tumour = np.sum(body == c_tumour)
    print(" ")
    print("Body")
    print("No. body blood cells",n_blood)
    print("No. body prostate cells",n_prostate)
    print("No. body tumour cells",n_tumour)
    print("No. body cells A",n_border + n_blood + n_prostate + n_tumour)
    n_surface = np.sum(body_steer == c_surface)
    n_duplicate = np.sum(body_steer == c_duplicate)
    n_exterior = np.sum(body_steer == c_exterior)
    n_end = np.sum(body_steer == c_end)
    n_neighbour_s = np.sum(body_steer == c_neighbour)
    print(" ")
    print("Steer")
    print("No. steer surface cells",n_surface)
    print("No. steer duplicate cells",n_duplicate)
    print("No. steer exterior cells",n_exterior)
    print("No. steer end cells",n_end)
    print("No. steer neighbour cells",n_neighbour)
    print("No. steer cells",(n_surface + n_duplicate + n_exterior + n_end +␣
 ↪n_neighbour))
    #
    return
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
```

```
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:35.423255

Date and time 2026-02-04 10:29:35.423895
Time since last check is 0:00:00.000640
```

1.5.2 Plot body data

Return to ToC: »

[3]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def plot_body(title, body_data, body_index, test_data, i_point, j_point,␣
 ↪k_point,
              plot_point, plot_blood, plot_prostate, plot_tumour, plot_surface,
              plot_duplicate, plot_exterior, plot_end, plot_neighbour,␣
 ↪plot_test,
              plot_2D, plot_log, show_legends,
              c_map, v_min = -1, v_max = -1):
    '''
    Plot body_data in 3D (and in 2D if required) with optional fixed limits on␣
 ↪the colour map.
    The data shown are selected using body_index.
    '''
    debug = False
    #
    # Markers for 3D and 2D plots
    m3_point = '*'
    m3_blood = '.'
    m3_prostate = 'o'
    m3_tumour = 'o'
    m3_surface = 'o'
    m3_duplicate = 'o'
    m3_exterior = 'o'
    m3_end = 'o'
    m3_neighbour = 'o'
    m3_test = '*'
    #
    m2_point  = '*'
    m2_blood = '.'
    m2_prostate = '+'
    m2_tumour = 'x'
    m2_surface = 's'
    m2_duplicate = 'D'
    m2_exterior = '.'
```

```
    m2_end = '^'
    m2_neighbour = 'v'
    m2_test = '*'
    #
    # Sizes for 3D and 2D plots
    s3_point = 20
    s3_blood = 0.001
    s3_prostate = 0.05
    s3_tumour = 5
    s3_surface = 5
    s3_duplicate = 20
    s3_exterior = 5
    s3_end = 20
    s3_neighbour = 20
    s3_test = 10
    #
    s2_point  = 20
    s2_blood = 0.05
    s2_prostate = 10
    s2_tumour = 10
    s2_surface = 10
    s2_duplicate = 20
    s2_exterior = 10
    s2_end = 20
    s2_neighbour = 20
    s2_test = 10
    #
    if plot_log:
        plot_data = np.log(np.maximum(body_data, 1e-6*np.ones_like(body_data)))
    else:
        plot_data = body_data
    #
    if debug:
        print(" ")
        print(f"Input v_min {v_min:.2f}, v_max {v_max:.2f}.")
    if v_min < 0:
        v_min = np.amin(plot_data)
    if v_max < 0:
        v_max = np.amax(plot_data)
    if v_min >= v_max:
        v_min = v_max - 0.1
    #
    if debug:
        print(" ")
        print(f"Used v_min {v_min:.2f}, v_max {v_max:.2f}.")
    #
    # Make 3D plot
```

```
    fig = plt.figure(figsize = (5, 7))
    #
    ax = fig.add_subplot(1, 1, 1, projection = '3d')
    ax.set_title(f"{title}, xyz")
    #
    if plot_blood:
        i_plot, j_plot, k_plot = np.where(body == c_blood)
        blood_body = plot_data[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_blood, s = s3_blood, vmin= v_min, vmax = v_max,
                   c = blood_body, alpha = 0.3, label = "Blood", cmap = c_map)
    #
    if plot_prostate:
        i_plot, j_plot, k_plot = np.where(body == c_prostate)
        prostate_body = plot_data[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_prostate, s = s3_prostate, vmin= v_min, vmax =␣
↪v_max,
                   c = prostate_body, alpha = 0.3, label = "Prostate", cmap =␣
↪c_map)
    #
    if plot_tumour:
        i_plot, j_plot, k_plot = np.where(body == c_tumour)
        tumour_body = plot_data[i_plot, j_plot, k_plot]
        im_t = ax.scatter(i_plot, j_plot, k_plot,
                          marker = m3_tumour, s = s3_tumour, vmin= v_min, vmax␣
↪= v_max,
                          c = tumour_body, alpha = 1.0, label = "Tumour", cmap␣
↪= c_map)
    #
    if plot_surface:
        i_plot, j_plot, k_plot = np.where(body_index == c_surface)
        surface_index = body_index[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_surface, s = s3_surface, vmin= v_min, vmax =␣
↪v_max,
                   c = surface_index, alpha = 1.0, label = "Surface", cmap =␣
↪c_map)
    #
    if plot_duplicate:
        i_plot, j_plot, k_plot = np.where(body_index == c_duplicate)
        dup_index = body_index[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_duplicate, s = s3_duplicate, vmin= v_min, vmax =␣
↪v_max,
```

```
                    c = dup_index, alpha = 1.0, label = "Duplicate", cmap =␣
↪c_map)
    #
    if plot_exterior:
        i_plot, j_plot, k_plot = np.where(body_index == c_exterior)
        exterior_index = body_index[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                    marker = m3_exterior, s = s3_exterior, vmin= v_min, vmax =␣
↪v_max,
                    c = exterior_index, alpha = 1.0, label = "Exterior", cmap =␣
↪c_map)
    #
    if plot_end:
        i_plot, j_plot, k_plot = np.where(body_index == c_end)
        end_index = body_index[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                    marker = m3_end, s = s3_end, vmin= v_min, vmax = v_max,
                    c = end_index, alpha = 1.0, label = "End", cmap = c_map)
    #
    if plot_neighbour:
        i_plot, j_plot, k_plot = np.where(body_index == c_neighbour)
        neigh_index = body_index[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                    marker = m3_neighbour, s = s3_neighbour, vmin= v_min, vmax =␣
↪v_max,
                    c = neigh_index, alpha = 1.0, label = "Neighbour", cmap =␣
↪c_map)
    #
    if plot_test:
        i_plot, j_plot, k_plot = np.where(body_index == c_test)
        test_use = test_data[i_plot, j_plot, k_plot]
        im_t = ax.scatter(i_plot, j_plot, k_plot,
                          marker = m3_test, s = s3_test, vmin= v_min, vmax =␣
↪v_max,
                          c = test_use, alpha = 1.0, label = title, cmap =␣
↪c_map)
    #
    if plot_point:
        ax.scatter(i_point, j_point, k_point,
                    marker = m3_point, s = s3_point,
                    color = 'r', alpha = 1.0, label = "Point")
    #
    ax.set_xlim(0, n_cells_x)
    ax.set_ylim(0, n_cells_y)
    ax.set_zlim(0, n_cells_z)
    ax.set_xlabel('x')
```

```
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    #
    if show_legends:
        ax.legend(loc = 'lower right')
    ax.zaxis.labelpad = 3.0
    #
    # Finish and display 3D plot
    fig.colorbar(im_t, ax = ax, shrink = 0.35, pad = 0.1)
    #
    plt.tight_layout()
    plt.show()
    #
    # Make 2D plots if required
    if plot_2D:
        fig = plt.figure(figsize = (7, 7))
        #
        ax_xy = fig.add_subplot(2, 2, 1)
        ax_xy.set_title(f"{title}, plane at k = {k_point}")
        #
        ax_xz = fig.add_subplot(2, 2, 2)
        ax_xz.set_title(f"{title}, plane at j = {j_point}")
        #
        ax_yz = fig.add_subplot(2, 2, 3)
        ax_yz.set_title(f"{title}, plane at i = {i_point}")
        #
        if plot_blood:
            i_plot, j_plot = np.where(body[:, :, k_point] == c_blood)
            colors = plot_data[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                          marker = m2_blood, s = s2_blood, vmin= v_min, vmax =␣
↪v_max,
                          c = colors, alpha = 1.0, label = "Blood", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body[:, j_point, :] == c_blood)
            colors = plot_data[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_blood, s = s2_blood, vmin= v_min, vmax =␣
↪v_max,
                          c = colors, alpha = 1.0, label = "Blood", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body[i_point, :, :] == c_blood)
            colors = plot_data[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
```

```
                        marker = m2_blood, s = s2_blood, vmin= v_min, vmax =␣
↪v_max,
                        c = colors, alpha = 1.0, label = "Blood", cmap =␣
↪c_map)
        #
        if plot_prostate:
            i_plot, j_plot = np.where(body[:, :, k_point] == c_prostate)
            colors = plot_data[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                        marker = m2_prostate, s = s2_prostate, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Prostate", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body[:, j_point, :] == c_prostate)
            colors = plot_data[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                        marker = m2_prostate, s = s2_prostate, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Prostate", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body[i_point, :, :] == c_prostate)
            colors = plot_data[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                        marker = m2_prostate, s = s2_prostate, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Prostate", cmap =␣
↪c_map)
        #
        if plot_tumour:
            i_plot, j_plot = np.where(body[:, :, k_point] == c_tumour)
            colors = plot_data[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                        marker = m2_tumour, s = s2_tumour, vmin= v_min, vmax␣
↪= v_max,
                        c = colors, alpha = 1.0, label = "Tumour", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body[:, j_point, :] == c_tumour)
            colors = plot_data[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                        marker = m2_tumour, s = s2_tumour, vmin= v_min, vmax␣
↪= v_max,
                        c = colors, alpha = 1.0, label = "Tumour", cmap =␣
↪c_map)
```

```
            #
            j_plot, k_plot = np.where(body[i_point, :, :] == c_tumour)
            colors = plot_data[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                          marker = m2_tumour, s = s2_tumour, vmin= v_min, vmax␣
↪= v_max,
                          c = colors, alpha = 1.0, label = "Tumour", cmap =␣
↪c_map)
        #
        if plot_surface:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_surface)
            colors = body_index[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                          marker = m2_surface, s = s2_surface, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Surface", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_surface)
            colors = body_index[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_surface, s = s2_surface, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Surface", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point,:, :] == c_surface)
            colors = body_index[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                          marker = m2_surface, s = s2_surface, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Surface", cmap =␣
↪c_map)
        #
        if plot_duplicate:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_duplicate)
            colors = body_index[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                          marker = m2_duplicate, s = s2_duplicate, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Duplicate", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_duplicate)
            colors = body_index[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
```

```
                        marker = m2_duplicate, s = s2_duplicate, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Duplicate", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point,:, :] == c_duplicate)
            colors = body_index[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                        marker = m2_duplicate, s = s2_duplicate, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Duplicate", cmap =␣
↪c_map)
        #
        if plot_exterior:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_exterior)
            colors = body_index[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                        marker = m2_exterior, s = s2_exterior, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Exterior", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_exterior)
            colors = body_index[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                        marker = m2_exterior, s = s2_exterior, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Exterior", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point,:, :] == c_exterior)
            colors = body_index[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                        marker = m2_exterior, s = s2_exterior, vmin= v_min,␣
↪vmax = v_max,
                        c = colors, alpha = 1.0, label = "Exterior", cmap =␣
↪c_map)
        #
        if plot_end:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_end)
            colors = body_index[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                        marker = m2_end, s = s2_end, vmin= v_min, vmax =␣
↪v_max,
                        c = colors, alpha = 1.0, label = "End", cmap = c_map)
            #
```

```
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_end)
            colors = body_index[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_end, s = s2_end, vmin= v_min, vmax =␣
↪v_max,
                          c = colors, alpha = 1.0, label = "End", cmap = c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point,:, :] == c_end)
            colors = body_index[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                          marker = m2_end, s = s2_end, vmin= v_min, vmax =␣
↪v_max,
                          c = colors, alpha = 1.0, label = "End", cmap = c_map)
        #
        if plot_neighbour:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_neighbour)
            colors = body_index[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                          marker = m2_neighbour, s = s2_neighbour, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Neighbour", cmap =␣
↪c_map)
            #
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_neighbour)
            colors = body_index[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_neighbour, s = s2_neighbour, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Neighbour", cmap =␣
↪c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point,:, :] == c_neighbour)
            colors = body_index[i_point, j_plot, k_plot]
            ax_yz.scatter(j_plot, k_plot,
                          marker = m2_neighbour, s = s2_neighbour, vmin= v_min,␣
↪vmax = v_max,
                          c = colors, alpha = 1.0, label = "Neighbour", cmap =␣
↪c_map)
        #
        if plot_test:
            i_plot, j_plot = np.where(body_index[:, :, k_point] == c_test)
            colors = test_data[i_plot, j_plot, k_point]
            ax_xy.scatter(i_plot, j_plot,
                          marker = m2_test, s = s2_test, vmin= v_min, vmax =␣
↪v_max,
                          c = colors, alpha = 1.0, label = title, cmap = c_map)
```

```
            #
            i_plot, k_plot = np.where(body_index[:, j_point, :] == c_test)
            colors = test_data[i_plot, j_point, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_test, s = s2_test, vmin= v_min, vmax =␣
 ↪v_max,
                          c = colors, alpha = 1.0, label = title, cmap = c_map)
            #
            j_plot, k_plot = np.where(body_index[i_point, :, :] == c_test)
            colors = test_data[i_point, j_plot, k_plot]
            ax_xz.scatter(i_plot, k_plot,
                          marker = m2_test, s = s2_test, vmin= v_min, vmax =␣
 ↪v_max,
                          c = colors, alpha = 1.0, label = title, cmap = c_map)
        #
        ax_xy.set_xlim(0, n_cells_x)
        ax_xy.set_ylim(0, n_cells_y)
        ax_xy.set_xlabel('x')
        ax_xy.set_ylabel('y')
        if show_legends:
            ax_xy.legend(loc = 'lower right')
        #
        ax_xz.set_xlim(0, n_cells_x)
        ax_xz.set_ylim(0, n_cells_z)
        ax_xz.set_xlabel('x')
        ax_xz.set_ylabel('z')
        if show_legends:
            ax_xz.legend(loc = 'lower right')
        #
        ax_yz.set_xlim(0, n_cells_y)
        ax_yz.set_ylim(0, n_cells_z)
        ax_yz.set_xlabel('y')
        ax_yz.set_ylabel('z')
        if show_legends:
            ax_yz.legend(loc = 'lower right')
        #
        plt.tight_layout()
        plt.show()
    #
    return
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:36.052509

Date and time 2026-02-04 10:29:36.054747
Time since last check is 0:00:00.002238
```

1.5.3 Set initial PSA levels

Return to ToC: »

[4]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def initialise_PSA(set_level, r_prop, with_plots, i_plot, j_plot, k_plot):
    '''
    Set initial PSA levels for body
    '''
    #
    debug = False
    with_plots = True
    #
    # For PSA values set 1
    if set_level == 2:
        init_fact = 1.0
    elif set_level == 1:
        init_fact = 0.5
    else:
        init_fact = 0.0001
    #
    # For PSA values set 2
    #init_fact *= 20
    #
    # PSA array
    PSA = np.zeros((n_is, n_js, n_ks))
    #
    # Set concentration in blood.
    n_blood = np.sum(body == c_blood)
    if set_level == 2:
        PSA[body == c_blood] = (0.8*init_fact*sig_prostate*np.ones(n_blood)*
                                (1 + r_prop*rng.random(n_blood)))
    elif set_level == 1:
        PSA[body == c_blood] = (0.1*init_fact*sig_prostate*np.ones(n_blood)*
                                (1 + r_prop*rng.random(n_blood)))
    else:
        PSA[body == c_blood] = 0.0
    #
    # Set concentration in prostate
    n_prostate = np.sum(body == c_prostate)
```

```
    PSA[body == c_prostate] = (init_fact*sig_prostate*np.ones(n_prostate)*
                               (1 + r_prop*rng.random(n_prostate)))
    #
    # Set concentration in tumour
    n_tumour = np.sum(body == c_tumour)
    PSA[body == c_tumour] = (init_fact*sig_tumour*np.ones(n_tumour)*
                             (1 + r_prop*rng.random(n_tumour)))
    #
    if debug:
        print(" ")
        print(f"Min initial PSA value {np.amin(PSA):.4f}")
        print(f"Max initial PSA value {np.amax(PSA):.4f}")
    #
    if with_plots:
        title = "PSA, t = 0"
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = True
        plot_prostate = True
        plot_tumour = True
        plot_surface = False
        plot_dup = False
        plot_ext = False
        plot_end = False
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        #c_map = cm.viridis
        c_map = cm.plasma
        #
        plot_body(title, PSA, body_steer, test_data, i_plot, j_plot, k_plot,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    return PSA
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
```

```
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:36.668371

Date and time 2026-02-04 10:29:36.668800
Time since last check is 0:00:00.000429
```

1.5.4 Set PSA level in borders

Return to ToC: »

1.5.5 Find exterior cells

Exterior cells are the blood cells immediately outside the protate and tumour.

Return to ToC: »

[5]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def find_exterior(with_plots):
    '''
    Find cells immediately outside prostate and tumour
    '''
    #
    # Find indices of cells in prostate and tumour
    i_pt, j_pt, k_pt = np.where(np.logical_or(body == c_prostate, body ==␣
 ↪c_tumour))
    #
    # Check not at border
    out_of_body = False
    if np.amax(i_pt) + 2 > n_is or np.amin(i_pt) < 2:
        print(f"Body limit reached in x: min(i_pt) = {np.amin(i_pt)}, max(i_pt)␣
 ↪= {np.amax(i_pt)}")
        out_of_body = True
    if np.amax(j_pt) + 2 > n_js or np.amin(j_pt) < 2:
        print(f"Body limit reached in y: min(j_pt) = {np.amin(j_pt)}, max(j_pt)␣
 ↪= {np.amax(j_pt)}")
        out_of_body = True
    if np.amax(k_pt) + 2 > n_ks or np.amin(k_pt) < 2:
        print(f"Body limit reached in z: min(k_pt) = {np.amin(k_pt)}, max(k_pt)␣
 ↪= {np.amax(k_pt)}")
        out_of_body = True
    if out_of_body:
        return  body_steer, out_of_body
    #
    n_new = np.sum(i_pt > 0)
    #
```

```
    # Find exterior cells and update flags in body_steer
    for n in range(0, n_new):
        indices = i_pt[n] - 1, j_pt[n], k_pt[n]
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
        indices = i_pt[n] + 1, j_pt[n], k_pt[n]
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
        indices = i_pt[n], j_pt[n] - 1, k_pt[n]
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
        indices = i_pt[n], j_pt[n] + 1, k_pt[n]
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
        indices = i_pt[n], j_pt[n], k_pt[n] - 1
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
        indices = i_pt[n], j_pt[n], k_pt[n] + 1
        if body[indices] == c_blood:
            body_steer[indices] = c_exterior
        #
    if with_plots:
        #
        # Make plots showing surface cells
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = False
        plot_dup = False
        plot_ext = True
        plot_end = False
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
```

```
        #
        title = f"Exterior, ({i_tumour}, {j_tumour}, {k_tumour})"
        plot_body(title, body_steer, body_steer, test_data, i_tumour, j_tumour,␣
 ↪k_tumour,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    return body_steer, out_of_body
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:37.702282

Date and time 2026-02-04 10:29:37.703089
Time since last check is 0:00:00.000807
```

1.5.6 Find area of blood/tumour interface

Identify "btcells", tumour cells in immediate contact with the blood.

Return to ToC: »

[6]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def area_tumour_blood(with_plots, i_time, i_plot, j_plot, k_plot):
    '''
    Find the area of the tumour/blood interface
    '''
    #
    # Find indices of cells in tumour
    i_t, j_t, k_t = np.where(body == c_tumour)
    #
    n_new = np.sum(i_t > 0)
    tot_area = 0.0
    #
    if with_plots:
        #
        # Markers for 3D and 2D plots
        m3_blood = '.'
        m3_prostate = 'o'
```

```
        m3_tumour = 'o'
        m3_test = 'o'
        #
        # Sizes for 3D and 2D plots
        s3_blood = 0.002
        s3_prostate = 0.05
        s3_tumour = 1
        s3_test = 10
        #
        #c_map_A = cm.viridis
        #c_map_B = cm.plasma
        c_map_A = cm.Blues
        c_map_B = cm.Reds
        #
        v_min_A = 0.0
        v_max_A = c_tumour
        v_min_B = 0.0
        v_max_B = 5.0
        #
        # Make 3D plot
        fig = plt.figure(figsize = (7, 7))
        #
        ax = fig.add_subplot(1, 1, 1, projection = '3d')
        ax.set_title(f"Interface area, time {i_time}")
        i_plot, j_plot, k_plot = np.where(body == c_blood)
        blood_body = body[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_blood, s = s3_blood, vmin= v_min_A, vmax =␣
↪v_max_A,
                   c = blood_body, alpha = 0.5, label = "Blood", cmap = c_map_A)
        #
        i_plot, j_plot, k_plot = np.where(body == c_prostate)
        prostate_body = body[i_plot, j_plot, k_plot]
        ax.scatter(i_plot, j_plot, k_plot,
                   marker = m3_prostate, s = s3_prostate, vmin= v_min_A, vmax =␣
↪v_max_A,
                   c = prostate_body, alpha = 0.5, label = "Prostate", cmap =␣
↪c_map_A)
        #
        i_plot, j_plot, k_plot = np.where(body == c_tumour)
        tumour_body = body[i_plot, j_plot, k_plot]
        im_b = ax.scatter(i_plot, j_plot, k_plot,
                          marker = m3_tumour, s = s3_tumour, vmin= v_min_A,␣
↪vmax = v_max_A,
                          c = tumour_body, alpha = 0.5, label = "Tumour", cmap␣
↪= c_map_A)
        #
```

```
        ax.set_xlim(0, n_cells_x)
        ax.set_ylim(0, n_cells_y)
        ax.set_zlim(0, n_cells_z)
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
    #
    # Find interface cells and number of faces in contact with blood cells.
    first_call = True
    for n in range(0, n_new):
        area = 0.0
        cell = i_t[n], j_t[n], k_t[n]
        #
        indices = i_t[n] - 1, j_t[n], k_t[n]
        if body[indices] == c_blood:
            area += 1.0
        #
        indices = i_t[n] + 1, j_t[n], k_t[n]
        if body[indices] == c_blood:
            area += 1.0
        #
        indices = i_t[n], j_t[n] - 1, k_t[n]
        if body[indices] == c_blood:
            area += 1.0
        #
        indices = i_t[n], j_t[n] + 1, k_t[n]
        if body[indices] == c_blood:
            area += 1.0
        #
        indices = i_t[n], j_t[n], k_t[n] - 1
        if body[indices] == c_blood:
            area += 1.0
        #
        indices = i_t[n], j_t[n], k_t[n] + 1
        if body[indices] == c_blood:
            area += 1.0
        #
        if with_plots and area > 0:
            if first_call:
                im_t = ax.scatter(i_t[n], j_t[n], k_t[n],
                                  marker = m3_test, s = s3_test, vmin =␣
↪v_min_B, vmax = v_max_B,
                                  c = area, alpha = 1.0, label = "Interface",␣
↪cmap = c_map_B)
                first_call = False
            else:
                im_t = ax.scatter(i_t[n], j_t[n], k_t[n],
```

```
                                    marker = m3_test, s = s3_test, vmin =␣
↪v_min_B, vmax = v_max_B,
                                    c = area, alpha = 1.0, cmap = c_map_B)
        #
        tot_area += area
        #
    btcell = tot_area > 0.0
    #
    if with_plots:
        #
        if show_legends:
            ax.legend(loc = 'lower right')
        ax.zaxis.labelpad = 3.0
        #
        # Finish and display 3D plot
        if first_call:
            fig.set_size_inches(5, 7)
            fig.colorbar(im_b, ax = ax, shrink = 0.35, pad = 0.1, location =␣
↪'right')
        else:
            fig.colorbar(im_b, ax = ax, shrink = 0.35, pad = 0.1, location =␣
↪'right')
            fig.colorbar(im_t, ax = ax, shrink = 0.35, location = 'left')
        #
        plt.tight_layout()
        plt.show()
    #
    return btcell, tot_area
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:38.367501

Date and time 2026-02-04 10:29:38.368221
Time since last check is 0:00:00.000720
```

1.5.7 Diffuse, create and decay PSA

Return to ToC: »

[7]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def do_diffusion(PSA, beta, r_prop, n_times, conv_test,
```

```
                with_init, set_level, with_all_plots, i_plot, j_plot, k_plot):
    '''
    Run diffusion for existing body.
    '''
    debug = False
    #
    #dt = 0.04 # PSA values set 1
    dt = 0.0001 # PSA values set 2
    nt_min = 2

    #nt_min = 20
    #
    if with_init:
        #
        # Initialise PSA here again so don't have to rerun above cell!
        with_plots = True
        PSA = initialise_PSA(set_level, r_prop, with_plots, i_plot, j_plot,␣
↪k_plot)
        #
    #
    # Choose times at which PSA distribution plotted
    make_plot = np.zeros(4).astype(int)
    make_plot[0] = 1
    make_plot[1] = n_times//16
    make_plot[2] = n_times//4
    make_plot[3] = n_times - 1
    #
    time = np.zeros(n_times)
    i_ptime = np.zeros(n_times).astype(int)
    n_tplots = 5
    tplot_i = np.zeros(n_tplots).astype(int)
    tplot_j = np.zeros(n_tplots).astype(int)
    tplot_k = np.zeros(n_tplots).astype(int)
    tplot_i[0], tplot_j[0], tplot_k[0] = i_plot, j_plot, k_plot
    tplot_i[1], tplot_j[1], tplot_k[1] = n_is//2, n_js//2, n_ks//2
    tplot_i[2], tplot_j[2], tplot_k[2] = 0, 0, 0
    tplot_i[3], tplot_j[3], tplot_k[3] = 1, 1, 1
    tplot_i[4], tplot_j[4], tplot_k[4] = n_is//4, n_js//4, n_ks//4
    #
    t_dev_ijk = np.zeros((n_times, n_tplots))
    t_dev_i = np.zeros((n_times, n_is))
    t_dev_j = np.zeros((n_times, n_js))
    t_dev_k = np.zeros((n_times, n_ks))
    #
    # Identify exterior cells
    with_plots_here = False
    body_steer, out_of_body = find_exterior(with_plots_here)
```

```
    if out_of_body:
        return PSA
    #
    # Do diffusion
    conv = False
    for nt in range(0, n_times):
        #
        i_ptime[nt] = nt
        time[nt] = nt*dt
        for tp in range(0, n_tplots):
            t_dev_ijk[nt, tp] = PSA[tplot_i[tp], tplot_j[tp], tplot_k[tp]]
        #
        t_dev_i[nt, :] = PSA[:, j_plot, k_plot]
        t_dev_j[nt, :] = PSA[i_plot, :, k_plot]
        t_dev_k[nt, :] = PSA[i_plot, j_plot, :]
        #
        # Do PSA diffusion
        delta_PSA = np.zeros((n_is, n_js, n_ks))
        #
        for i in range(1, n_cells_x - 1):
            for j in range(1, n_cells_y - 1):
                for k in range(1, n_cells_z - 1):
                    #
                    # Calculate PSA change for prostate, tumour, "exterior" blood
                    if (body[i, j, k] == c_prostate or
                        body[i, j, k] == c_tumour or
                        body_steer[i, j, k] == c_exterior):
                        delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1, j, k] +
                                                      PSA[i, j - 1, k] + PSA[i, j + 1, k] +
                                                      PSA[i, j, k - 1] + PSA[i, j, k + 1] -
                                                      6*PSA[i, j, k])
                    #
                #
            #
        # Deal with faces
        i = 0
        for j in range(1, n_cells_y - 1):
            for k in range(1, n_cells_z - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
```

```
                                                PSA[i, j - 1, k] + PSA[i, j +␣
↪1, k] +
                                                PSA[i, j, k - 1] + PSA[i, j,␣
↪k + 1] -
                                                5*PSA[i, j, k])
                #
            #
        i = n_cells_x - 1
        for j in range(1, n_cells_y - 1):
            for k in range(1, n_cells_z - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                                PSA[i, j - 1, k] + PSA[i, j +␣
↪1, k] +
                                                PSA[i, j, k - 1] + PSA[i, j,␣
↪k + 1] -
                                                5*PSA[i, j, k])
                #
            #
        j = 0
        for i in range(1, n_cells_x - 1):
            for k in range(1, n_cells_z - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1,␣
↪j, k] +
                                                PSA[i, j + 1, k] +
                                                PSA[i, j, k - 1] + PSA[i, j,␣
↪k + 1] -
                                                5*PSA[i, j, k])
                #
            #
        j = n_cells_x - 1
        for i in range(1, n_cells_x - 1):
            for k in range(1, n_cells_z - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1,␣
↪j, k] +
                                                PSA[i, j - 1, k] +
                                                PSA[i, j, k - 1] + PSA[i, j,␣
↪k + 1] -
```

```
                                                5*PSA[i, j, k])
                #
        k = 0
        for i in range(1, n_cells_x - 1):
            for j in range(1, n_cells_y - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1,␣
↪j, k] +
                                                  PSA[i, j - 1, k] + PSA[i, j +␣
↪1, k] +
                                                  PSA[i, j, k + 1] -
                                                  5*PSA[i, j, k])
                #
            #
        k = n_cells_z - 1
        for i in range(1, n_cells_x - 1):
            for j in range(1, n_cells_y - 1):
                if (body[i, j, k] == c_prostate or
                    body[i, j, k] == c_tumour or
                    body_steer[i, j, k] == c_exterior):
                    delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1,␣
↪j, k] +
                                                  PSA[i, j - 1, k] + PSA[i, j +␣
↪1, k] +
                                                  PSA[i, j, k - 1] -
                                                  5*PSA[i, j, k])
                #
            #
        # Deal with edges
        i = 0
        j = 0
        for k in range(1, n_cells_z - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                              PSA[i, j + 1, k] +
                                              PSA[i, j, k - 1] + PSA[i, j, k +␣
↪1] -
                                              4*PSA[i, j, k])
            #
        i = 0
        j = n_cells_y - 1
        for k in range(1, n_cells_z - 1):
            if (body[i, j, k] == c_prostate or
```

```
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                              PSA[i, j - 1, k] +
                                              PSA[i, j, k - 1] + PSA[i, j, k +␣
↪1] -
                                              4*PSA[i, j, k])
            #
        i = n_cells_x - 1
        j = 0
        for k in range(1, n_cells_z - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                              PSA[i, j + 1, k] +
                                              PSA[i, j, k - 1] + PSA[i, j, k +␣
↪1] -
                                              4*PSA[i, j, k])
            #
        i = n_cells_x - 1
        j = n_cells_y - 1
        for k in range(1, n_cells_z - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                              PSA[i, j + 1, k] +
                                              PSA[i, j, k - 1] + PSA[i, j, k +␣
↪1] -
                                              4*PSA[i, j, k])
            #
        i = 0
        k = 0
        for j in range(1, n_cells_y - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                              PSA[i, j - 1, k] + PSA[i, j + 1,␣
↪k] +
                                              PSA[i, j, k + 1] -
                                              4*PSA[i, j, k])
            #
        i = 0
        k = n_cells_z - 1
        for j in range(1, n_cells_y - 1):
```

```
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                              PSA[i, j - 1, k] + PSA[i, j + 1,␣
↪k] +
                                              PSA[i, j, k - 1] -
                                              6*PSA[i, j, k])
            #
        i = n_cells_x - 1
        k = 0
        for j in range(1, n_cells_y - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                              PSA[i, j - 1, k] + PSA[i, j + 1,␣
↪k] +
                                              PSA[i, j, k + 1] -
                                              4*PSA[i, j, k])
            #
        i = n_cells_x - 1
        k = 0
        for j in range(1, n_cells_y - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                              PSA[i, j - 1, k] + PSA[i, j + 1,␣
↪k] +
                                              PSA[i, j, k - 1] -
                                              6*PSA[i, j, k])
            #
        j = 0
        k = 0
        for i in range(1, n_cells_x - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1, j,␣
↪k] +
                                              PSA[i, j + 1, k] +
                                              PSA[i, j, k + 1] -
                                              4*PSA[i, j, k])
            #
        j = 0
        k = n_cells_z - 1
```

```
        for i in range(1, n_cells_x - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1, j,␣
↪k] +
                                              PSA[i, j + 1, k] +
                                              PSA[i, j, k - 1] -
                                              6*PSA[i, j, k])
            #
        j = n_cells_y - 1
        k = 0
        for i in range(1, n_cells_x - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1, j,␣
↪k] +
                                              PSA[i, j - 1, k] +
                                              PSA[i, j, k + 1] -
                                              4*PSA[i, j, k])
            #
        j = n_cells_y - 1
        k = n_cells_z - 1
        for i in range(1, n_cells_x - 1):
            if (body[i, j, k] == c_prostate or
                body[i, j, k] == c_tumour or
                body_steer[i, j, k] == c_exterior):
                delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] + PSA[i + 1, j,␣
↪k] +
                                              PSA[i, j - 1, k] +
                                              PSA[i, j, k - 1] -
                                              4*PSA[i, j, k])
            #
        # Deal with corners
        i = 0
        j = 0
        k = 0
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                          PSA[i, j + 1, k] +
                                          PSA[i, j, k + 1] -
                                          3*PSA[i, j, k])
        #
        i = 0
```

```
        j = 0
        k = n_cells_z - 1
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                          PSA[i, j + 1, k] +
                                          PSA[i, j, k - 1] -
                                          3*PSA[i, j, k])
        #
        i = 0
        j = n_cells_y - 1
        k = 0
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                          PSA[i, j - 1, k] +
                                          PSA[i, j, k + 1] -
                                          3*PSA[i, j, k])
        #
        i = 0
        j = n_cells_y - 1
        k = n_cells_z - 1
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i + 1, j, k] +
                                          PSA[i, j - 1, k] +
                                          PSA[i, j, k - 1] -
                                          3*PSA[i, j, k])
        #
        i = n_cells_x - 1
        j = 0
        k = 0
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                          PSA[i, j + 1, k] +
                                          PSA[i, j, k + 1] -
                                          3*PSA[i, j, k])
        #
        i = n_cells_x - 1
        j = 0
        k = n_cells_z - 1
        if (body[i, j, k] == c_prostate or
```

```
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                          PSA[i, j + 1, k] +
                                          PSA[i, j, k - 1] -
                                          3*PSA[i, j, k])
        #
        i = n_cells_x - 1
        j = n_cells_y - 1
        k = 0
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                          PSA[i, j - 1, k] +
                                          PSA[i, j, k + 1] -
                                          3*PSA[i, j, k])
        #
        i = n_cells_x - 1
        j = n_cells_y - 1
        k = n_cells_z - 1
        if (body[i, j, k] == c_prostate or
            body[i, j, k] == c_tumour or
            body_steer[i, j, k] == c_exterior):
            delta_PSA[i, j, k] = dt*beta*(PSA[i - 1, j, k] +
                                          PSA[i, j - 1, k] +
                                          PSA[i, j, k - 1] -
                                          3*PSA[i, j, k])
            #
        # Find average of change in exterior PSA level
        ext_inds = np.where(body_steer == c_exterior)
        avg_new_PSA = np.average(delta_PSA[ext_inds])
        #
        # Set all blood changes to average external PSA value
        delta_PSA[body == c_blood] = avg_new_PSA
        #
        # Do PSA decay
        n_blood = np.sum(body == c_blood)
        delta_PSA[body == c_blood] -= dt/tau_blood*PSA[body == c_blood]
        #
        n_prostate = np.sum(body == c_prostate)
        delta_PSA[body == c_prostate] -= dt/tau_prostate*PSA[body == c_prostate]
        #
        n_tumour = np.sum(body == c_tumour)
        delta_PSA[body == c_tumour] -= dt/tau_tumour*PSA[body == c_tumour]
        #
        # Do PSA creation
```

```
        delta_PSA[body == c_prostate] += \
            dt*sig_prostate*np.ones(n_prostate)*(1 + r_prop*rng.
↪random(n_prostate))
        #
        delta_PSA[body == c_tumour] += \
            dt*sig_tumour*np.ones(n_tumour)*(1 + r_prop*rng.random(n_tumour))
        #
        PSA = PSA + delta_PSA
        #
        # Interim plot
        if nt in make_plot and with_all_plots:
            if debug:
                print(" ")
                print(f"Min PSA value {np.amin(PSA):.4f}")
                print(f"Max PSA value {np.amax(PSA):.4f}")
            #
            # Steer plotting
            plot_point = False
            plot_blood = True
            plot_prostate = True
            plot_tumour = True
            plot_surface = False
            plot_dup = False
            plot_ext = False
            plot_end = False
            plot_neigh = False
            plot_test = False
            #
            #plot_log = True
            plot_log = False
            plot_2D = True
            show_legends = True
            #
            c_map = cm.plasma
            #v_min = -6.0
            #v_max = 1.5
            v_min = -1.0
            v_max = -1.0
            #
            title = f"PSA, t = {nt}"
            plot_body(title, PSA, body_steer, test_data, i_plot, j_plot, k_plot,
                      plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
                      plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                      plot_2D, plot_log, show_legends,
                      c_map, v_min = v_min, v_max = v_max)
            #
```

```
        ext_test = abs(np.amax(delta_PSA[ext_inds]/PSA[ext_inds]))
        inds_here = np.logical_or(body == c_prostate, body == c_tumour)
        prostate_test = abs(np.amax(delta_PSA[inds_here]/PSA[inds_here]))
        epsilon = 1e-28
        if (epsilon < ext_test < conv_test and
            epsilon < prostate_test < conv_test and
            nt > nt_min):
            print(f"Diffusion converged, nt = {nt}, ext_test = {ext_test:.6e}, "
                  f"prostate_test = {prostate_test:.6e}")
            conv = True
            break
        #
    if not conv:
        print(f"Diffusion didn't converge, nt = {nt}, ext_test = {ext_test:.
↪6e}, "
              f"prostate_test = {prostate_test:.6e}")
        #
    title = f"PSA, t = {nt}"
    plot_point = False
    plot_blood = True
    plot_prostate = True
    plot_tumour = True
    plot_surface = False
    plot_dup = False
    plot_ext = False
    plot_end = False
    plot_neigh = False
    plot_test = False
    #
    plot_2D = True
    #plot_log = True
    plot_log = False
    show_legends = True
    #
    c_map = cm.plasma
    #v_min = -6.0
    #v_max = 1.5
    v_min = -1.0
    v_max = -1.0
    #
    if debug:
        print(" ")
        print(f"Min PSA value {np.amin(PSA):.4f}")
        print(f"Max PSA value {np.amax(PSA):.4f}")
    #
    plot_body(title, PSA, body_steer, test_data, i_plot, j_plot, k_plot,
              plot_point, plot_blood, plot_prostate, plot_tumour, plot_surface,
```

```
                plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                plot_2D, plot_log, show_legends,
                c_map, v_min = v_min, v_max = v_max)
    #
    i_ptime = i_ptime[0:nt]
    tplot_i = tplot_i[0:nt]
    tplot_j = tplot_j[0:nt]
    tplot_k = tplot_k[0:nt]
    t_dev_ijk = t_dev_ijk[0:nt, :]
    t_dev_i = t_dev_i[0:nt, :]
    t_dev_j = t_dev_j[0:nt, :]
    t_dev_k = t_dev_k[0:nt, :]
    #
    plot_time_dev(i_ptime, tplot_i, tplot_j, tplot_k, t_dev_ijk, t_dev_i,␣
 ↪t_dev_j, t_dev_k,
                  i_plot, j_plot, k_plot)
    #
    return PSA
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:39.077960

Date and time 2026-02-04 10:29:39.081066
Time since last check is 0:00:00.003106
```

1.5.8 Plot time development of PSA levels

The "time" here is that required for the PSA levels to reach a stable(ish) state.

Return to ToC: »

[8]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def plot_time_dev(time, tplot_i, tplot_j, tplot_k, t_dev_ijk, t_dev_i, t_dev_j,␣
 ↪t_dev_k,
                  i_plot, j_plot, k_plot):
    '''
    Plot time development of PSA concentrations at some specified points, given␣
 ↪in t_dev_ijk, and
    along lines defined by i_plot, j_plot and k_plot.
    Make sure the maximum concentrations are in the zeroth component of␣
 ↪t_dev_ijk if you want
```

```
    to use the scaling that comes when plot_log = False!
    '''
    #
    debug = False
    #
    plot_log = True
    #
    # Set times at which lines in i ,j, and k are displayed
    plot_time_A = min(5, np.amax(time).astype(int))
    plot_step_A = 1
    plot_time_B = min(30, np.amax(time).astype(int))
    plot_step_B = 10
    plot_time_C = np.amax(time).astype(int)
    plot_step_C = max(np.amax(time).astype(int)//10, 1)
    #
    # Make plots at specified points
    n_tplots = len(tplot_i)
    #
    fig = plt.figure(figsize = (6, 9))
    #
    ax = fig.add_subplot(4, 1, 1)
    ax.set_title(f"PSA with t at specified points")
    ax.set_xlabel("t")
    if plot_log:
        ax.set_ylabel("log(PSA)")
        for tp in range(0, n_tplots):
            bool_plot = t_dev_ijk[:, tp] > 0
            ax.plot(time[bool_plot], np.log(t_dev_ijk[bool_plot, tp]),
                    label = f"({tplot_i[tp]}, {tplot_j[tp]}, {tplot_k[tp]})")
    else:
        max_zero = np.amax(t_dev_ijk[:, 0])
        max_other = np.amax(t_dev_ijk[:,1:n_tplots])
        max_ratio = max_zero/max_other
        ax.set_ylabel("PSA")
        for tp in range(0, n_tplots):
            scale = 1 + 0.2*max_ratio*(tp > 0)
            ax.plot(time[:], scale*t_dev_ijk[:, tp],
                    label = f"scale {scale:.0f}, ({tplot_i[tp]}, {tplot_j[tp]},␣
↪{tplot_k[tp]})")
    ax.grid(color = 'g')
    ax.legend(loc = "upper right")
    #
    # Make plot at fixed i
    plot_log = True
    #
    cell_x = np.linspace(0, n_is - 1, n_is)
    cell_y = np.linspace(0, n_js - 1, n_js)
```

```
    cell_z = np.linspace(0, n_ks - 1, n_ks)
    #
    ax = fig.add_subplot(4, 1, 2)
    ax.set_title(f"PSA with t along x at i = {i_plot}")
    ax.set_xlabel("x")
    ax.set_ylabel("PSA")
    for nt in range(0, plot_time_A, plot_step_A):
        ax.plot(cell_x[:], t_dev_i[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_A, plot_time_B, plot_step_B):
        ax.plot(cell_x[:], t_dev_i[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_B, plot_time_C, plot_step_C):
        if nt == plot_time_B:
            ax.plot(cell_x[:], t_dev_i[nt, :], label = f"Others, t $\\geq$␣
↪{nt}")
        else:
            ax.plot(cell_x[:], t_dev_i[nt, :])
    if plot_log:
        ax.set_yscale('log')
    ax.grid(color = 'g')
    ax.legend()
    #
    # Make plot at fixed j
    ax = fig.add_subplot(4, 1, 3)
    ax.set_title(f"PSA with time along y at j =  {j_plot}")
    ax.set_xlabel("y")
    ax.set_ylabel("PSA")
    for nt in range(0, plot_time_A, plot_step_A):
        ax.plot(cell_y[:], t_dev_j[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_A, plot_time_B, plot_step_B):
        ax.plot(cell_y[:], t_dev_j[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_B, plot_time_C, plot_step_C):
        if nt == plot_time_B:
            ax.plot(cell_y[:], t_dev_j[nt, :], label = f"Others, t $\\geq$␣
↪{nt}")
        else:
            ax.plot(cell_y[:], t_dev_j[nt, :])
    if plot_log:
        ax.set_yscale('log')
    ax.grid(color = 'g')
    ax.legend()
    #
    # Make plot at fixed k
    ax = fig.add_subplot(4, 1, 4)
    ax.set_title(f"PSA with t along z at k = {k_plot}")
    ax.set_xlabel("z")
    ax.set_ylabel("PSA")
    for nt in range(0, plot_time_A, plot_step_A):
```

```
        ax.plot(cell_z[:], t_dev_k[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_A, plot_time_B, plot_step_B):
        ax.plot(cell_z[:], t_dev_k[nt, :], label = f"t = {nt}")
    for nt in range(plot_time_B, plot_time_C, plot_step_C):
        if nt == plot_time_B:
            ax.plot(cell_z[:], t_dev_k[nt, :], label = f"Others, t $\\geq$␣
 ↪{nt}")
        else:
            ax.plot(cell_z[:], t_dev_k[nt, :])
        ax.plot(cell_z[:], t_dev_k[nt, :])
    if plot_log:
        ax.set_yscale('log')
    ax.grid(color = 'g')
    ax.legend()
    #
    plt.tight_layout()
    plt.show()
    #
    return
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:39.589601

Date and time 2026-02-04 10:29:39.590555
Time since last check is 0:00:00.000954
```

1.5.9 Choose position for duplicate cell

Return to ToC: »

[9]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def find_duplicator(with_plots):
    '''
    Find one of cells in tumour surface to serve as position of duplicate
    '''
    debug = False
    #
    # Check whether there is already a duplicate (can only do one at a time!)
    if np.sum(body_steer == c_duplicate) > 0:
        print(" ")
        print("Found pre-existing duplicate in find_duplicator - stop")
```

```
        sys.exit(0)
        #
    duplicate = np.zeros(3).astype(int)
    #
    # Find indices of cells in tumour
    out_of_body = False
    i_t, j_t, k_t = np.where(body == c_tumour)
    if np.amax(i_t) + 2 > n_is or np.amin(i_t) < 2:
        print(f"Body limit reached in x: min(i_t) = {np.amin(i_t)}, max(i_t) =␣
↪{np.amax(i_t)}")
        out_of_body = True
    if np.amax(j_t) + 2 > n_js or np.amin(j_t) < 2:
        print(f"Body limit reached in y: min(j_t) = {np.amin(j_t)}, max(j_t) =␣
↪{np.amax(j_t)}")
        out_of_body = True
    if np.amax(k_t) + 2 > n_ks or np.amin(k_t) < 2:
        print(f"Body limit reached in z: min(k_t) = {np.amin(k_t)}, max(k_t) =␣
↪{np.amax(k_t)}")
        out_of_body = True
    #
    if out_of_body:
        return duplicate, body_steer, out_of_body
    #
    n_t = len(i_t)
    #
    # Find surface cells and update flags in body_steer array
    for n in range(0, n_t):
        indices = i_t[n] - 1, j_t[n], k_t[n]
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
        indices = i_t[n] + 1, j_t[n], k_t[n]
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
        indices = i_t[n], j_t[n] - 1, k_t[n]
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
        indices = i_t[n], j_t[n] + 1, k_t[n]
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
        indices = i_t[n], j_t[n], k_t[n] - 1
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
```

```
        indices = i_t[n], j_t[n], k_t[n] + 1
        if body[indices] != c_tumour:
            body_steer[indices] = c_surface
        #
    if with_plots:
        #
        # Make plots showing surface cells
        v_min = -1.0
        v_max = -1.0
        #
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
        plot_dup = False
        plot_ext = False
        plot_end = False
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"Surface ({i_tumour}, {j_tumour}, {k_tumour})"
        plot_body(title, body_steer, body_steer, test_data, i_tumour, j_tumour,␣
↪k_tumour,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    # Randomly choose surface cell for duplication
    surface = np.where(body_steer == c_surface)
    i_surface = surface[0]
    j_surface = surface[1]
    k_surface = surface[2]
    n_surface = len(i_surface)
    ind_dup = rng.integers(0, high = n_surface, size = 1)[0]
    i_dup = i_surface[ind_dup]
    j_dup = j_surface[ind_dup]
    k_dup = k_surface[ind_dup]
    #
```

```
    duplicate[0] = i_dup
    duplicate[1] = j_dup
    duplicate[2] = k_dup
    #
    if debug:
        print(" ")
        print("i_t \n",i_t)
        print("j_t \n",j_t)
        print("k_t \n",k_t)
        print(f"i_surface {i_surface}, j_surface {j_surface}, k_surface␣
↪{k_surface}")
        print(f"n_surface {n_surface}")
        print(f"ind_dup {ind_dup}")
        print(f"i_dup {i_dup}, j_dup {j_dup}, k_dup {k_dup}")
        print(f"Duplicate body type {body[i_dup, j_dup, k_dup]}")
    #
    # Keep this below the debug printout above or you will get confused as␣
↪c_duplicate
    # overwrites one of the surface values!
    body_steer[i_dup, j_dup, k_dup] = c_duplicate
    #
    if with_plots:
        #
        # Make plot showing the cell to be duplicated
        v_min = -1.0
        v_max = -1.0
        #
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
        plot_dup = True
        plot_ext = False
        plot_end = False
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"Duplicate ({i_dup}, {j_dup}, {k_dup})"
        plot_body(title, body_steer, body_steer, test_data, i_tumour, j_tumour,␣
↪k_tumour,
```

```
                    plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                    plot_dup, plot_ext, plot_end, plot_neigh,plot_test,
                    plot_2D, plot_log, show_legends,
                    c_map, v_min = v_min, v_max = v_max)
        #
    #
    return duplicate, body_steer, out_of_body
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:40.192814

Date and time 2026-02-04 10:29:40.194410
Time since last check is 0:00:00.001596
```

1.5.10 Show all changes in body

Return to ToC: »

[10]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def show_change(body_org, i_plot, j_plot, k_plot, with_plots, title):
    '''
    Plot changes due to duplication(s). Highlight changes between body_org and␣
 ↪current body.
    '''
    debug = False
    #
    body_change = np.zeros((n_is, n_js, n_ks))
    #
    # Label changes as test
    body_change[body != body_org] = c_test
    if debug:
        print(" ")
        print("np.where(body_change == c_test)",np.where(body_change == c_test))
    #
    i_change, j_change, k_change = np.where(body_change == c_test)
    if debug:
        print(" ")
        print(f"Number of body changes detected {int(np.sum(body_change ==␣
 ↪c_test))}")
        print(f"Change i min {np.amin(i_change)}, i max {np.amax(i_change)}")
```

```
        print(f"Change j min {np.amin(j_change)}, j max {np.amax(j_change)}")
        print(f"Change k min {np.amin(j_change)}, k max {np.amax(j_change)}")
        print(" ")
        print(f"Change x min {cell_x[np.amin(i_change)]}, x max {cell_x[np.
↪amax(i_change)]}")
        print(f"Change y min {cell_y[np.amin(j_change)]}, y max {cell_y[np.
↪amax(j_change)]}")
        print(f"Change z min {cell_z[np.amin(k_change)]}, z max {cell_z[np.
↪amax(k_change)]}")
        print(" ")
        print(f"Prostate centre, indices {i_prostate}, {j_prostate},␣
↪{k_prostate}")
        print(f"Prostate radius {prostate_rad}")
        print(" ")
        print(f"Tumour centre, indices {i_tumour}, {j_tumour}, {k_tumour}")
        print(f"Tumour radius {tumour_rad}")
    #
    # Make plot showing changes due to duplication
    if with_plots:
        print("")
        v_min = -1.0
        v_max = -1.0
        #
        plot_point = True
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = False
        plot_dup = False
        plot_ext = False
        plot_end = False
        plot_neigh = False
        plot_test = True
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        plot_body(title, body_change, body_change, test_data, i_plot, j_plot,␣
↪k_plot,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
```

```
                    c_map, v_min = v_min, v_max = v_max)
    return
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:40.814733

Date and time 2026-02-04 10:29:40.816047
Time since last check is 0:00:00.001314
```

1.5.11 Find new cell positions

One of surface cells has been chosen to be the position of the duplicate. The situation before shifting cells is as illustrated in **Figure 2**.

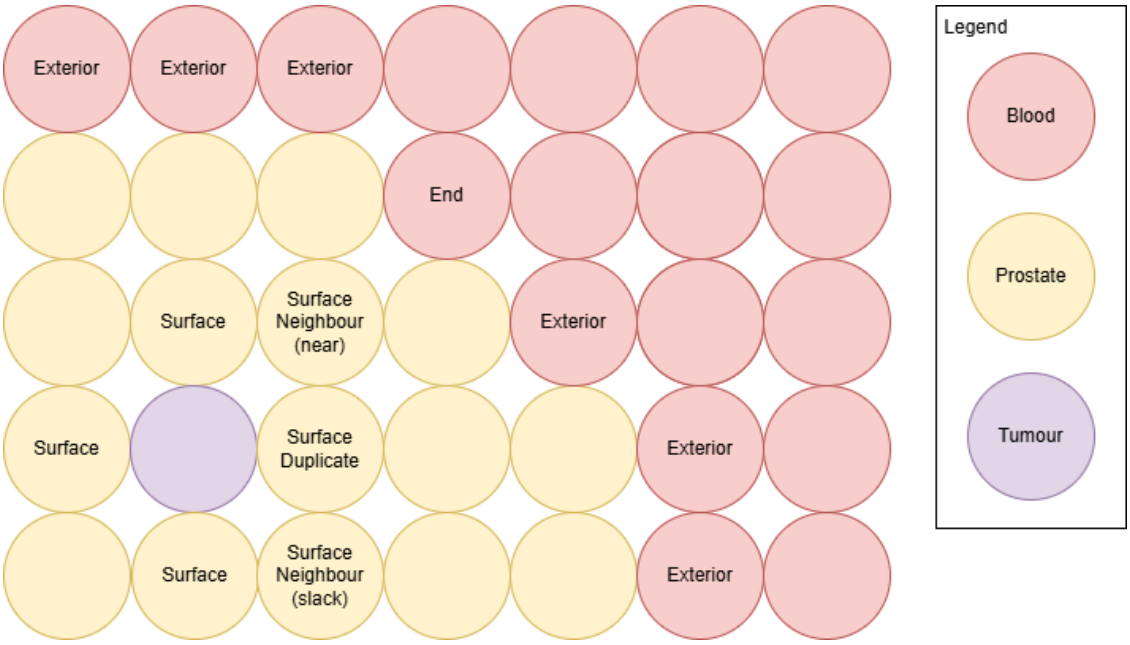


Figure 2 *Location of duplicate, surface, exterior, neighbour and end cells before shifting operation;* `position_finder_near` *ensures that the neighbour cell is as close to the end cell as possible, while* `position_finder_slack` *does not have this requirememt.*

In the `position_finder_near` routine, the "neighbour" is chosen to be as close as possible to the "end" cell. An attempt is first made to find a suitable cell which is not a tumour cell. If this is impossible, a tumour cell may be chosen as a neighbour. This is done as the neighbour is moved to a position outside the tumour and prostate, and it is possible that this means a tumour cell is disconnected from the tumour if the neighbour it a tumour cell.

In `position_finder_slack`, the neighbour may be anywhere around the original tumour cell, with no reference to the end cell position. An attempt is first made to find a suitable cell which is not a tumour cell. If this is impossible, a tumour cell may be chosen as a neighbour. This is done as

the neighbour is moved to a position outside the tumour and prostate, and it is possible that this means a tumour cell is disconnected from the tumour if the neighbour is a tumour cell. As the neighbour can be further from the end cell when `position_finder_slack` is used, the resulting tumour is more widely spread.

Situation after shifting cells:

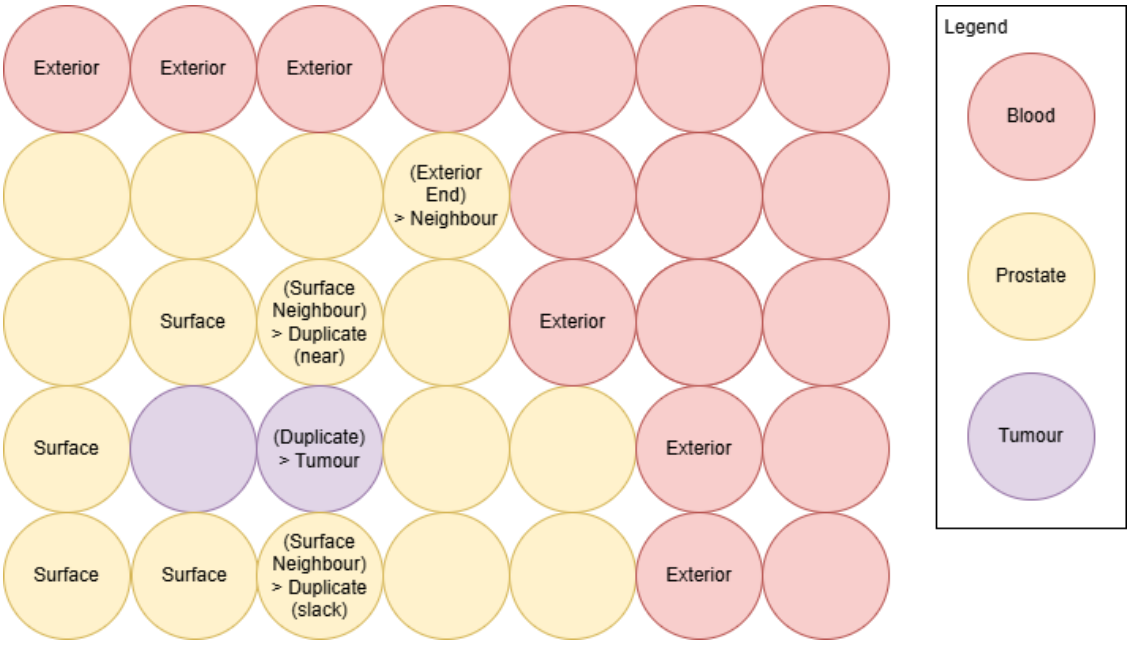


Figure 2 *Location of new tumour, shifted duplicate and neighbour cells after shifting operation.*

Near position finder Return to ToC: »

[11]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def position_finder_near(body_steer, i_dup, j_dup, k_dup, with_plots):
    '''
    Find the "neighbour" cell position to accommodate the "duplicate" and the␣
 ↪"end" position
    to accommodate the neighbour. The end is a position outside the prostate␣
 ↪and tumour.
    In this version, the end cell is used is that closest to the duplicate.
    An attempt is first made to find a neighbour that is not part of the tumour
    and after that, tumour cells are allowed to be neighbours.
    '''
    debug = False
    #
    neighbour = np.zeros(3).astype(int)
    end_point = np.zeros(3).astype(int)
    #
```

```
    # Check whether there are too many duplicates (can only do one at a time!)
    n_duplicates = np.sum(body_steer == c_duplicate)
    if n_duplicates > 1:
        print(" ")
        print(f"Found {n_duplicates} duplicates in position_finder - stop")
        sys.exit(0)
    #
    # Find distances from cell that will accept duplication to exterior cells. The closest
    # exterior cell is chosen as the "end" cell. If find_exterior has already been run,
    # don't do it again!
    n_ext = np.sum(body_steer == c_exterior)
    if n_ext < 1:
        body_steer, out_of_body = find_exterior(with_plots)
        if out_of_body:
            return body_steer, neighbour, end_point, out_of_body
    #
    n_ext = np.sum(body_steer == c_exterior)
    d_to_e_sq = np.zeros(n_ext)
    i_d_to_e, j_d_to_e, k_d_to_e = np.where(body_steer == c_exterior)
    d_to_e_sq = (i_dup - i_d_to_e)**2 + (j_dup - j_d_to_e)**2 + (k_dup - k_d_to_e)**2
    #
    # Find indices in dup_to_ext_sq that give minimum distance
    min_dist_ind = np.argsort(d_to_e_sq)
    #
    # Randomly order min_dist_ind where there are groups of distances that are the same
    # (Not sure that this is necessary!)
    new_dist_ind = np.zeros(n_ext).astype(int)
    #
    un_dist_array, un_dist_ind, n_un_dist = np.unique(d_to_e_sq, return_index = True,
                                                      return_inverse = False,
                                                      return_counts = True)
    n_groups = len(n_un_dist)
    n_bot = 0
    for n in range(0, n_groups):
        n_top = n_bot + n_un_dist[n]
        inds_here = np.zeros(n_un_dist[n]).astype(int)
        inds_here[0:n_un_dist[n]] = min_dist_ind[n_bot:n_top]
        rng.shuffle(inds_here)
        new_dist_ind[n_bot:n_top] = inds_here[0:n_un_dist[n]]
        n_bot = n_top
    #
    # Use below rather than above if don't want to do explicit random ordering!
```

```
    #new_dist_ind = min_dist_ind
    #
    # Define "end" cell, checking that it is not same as duplicate.
    duplicate = np.array([i_dup, j_dup, k_dup]).astype(int)
    origin = np.zeros(3).astype(int)
    end_test = np.zeros(3).astype(int)
    if debug:
        print(" ")
        print("Find end point")
        print("No. in exterior, len(min_dist_ind)",len(min_dist_ind))
        print("i_dup, j_dup, k_dup",i_dup, j_dup, k_dup)
    #
    w = 0
    out_of_body = False
    while w < len(new_dist_ind) and np.all(end_point == origin):
        min_ind = new_dist_ind[w]
        end_test[0] = i_d_to_e[min_ind]
        end_test[1] = j_d_to_e[min_ind]
        end_test[2] = k_d_to_e[min_ind]
        if debug:
            print(f"w {w}, end_point {end_point}, end_test {end_test}")
        if not np.all(end_test == duplicate):
            min_d_to_e = min_ind
            end_point[:] = end_test[:]
        w += 1
    if np.all(end_point == origin):
        print(" ")
        print("No end point found - stop")
        sys.exit(0)
    if debug:
        print(" ")
        print(f"duplicate {duplicate}")
        print(f"end_point {end_point}")
    #
    i_end, j_end, k_end = end_point[0], end_point[1], end_point[2]
    if i_end > n_is - 2 or i_end < 2:
        print(" ")
        print(f"End point out of body, i_end {i_end}")
        out_of_body = True
    if j_end > n_js - 2 or j_end < 2:
        print(" ")
        print(f"End point out of body, j_end {j_end}")
        out_of_body = True
    if k_end > n_ks - 2 or k_end < 2:
        print(" ")
        print(f"End point out of body, k_end {k_end}")
        out_of_body = True
```

```
    if out_of_body:
        return body_steer, neighbour, end_point, out_of_body
    #
    body_steer[i_end, j_end, k_end] = c_end
    #
    if with_plots:
        #
        # Make plot showing the cell to be duplicated and end cell
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
        plot_dup = True
        plot_ext = True
        plot_end = True
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"End ({i_end}, {j_end}, {k_end})"
        plot_body(title, body_steer, body_steer, test_data, i_end, j_end, k_end,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    # Choose "neighbour" cell to shift to end position. Cell adjacent to␣
↪duplicate can be prostate,
    # tumour or blood. Will be shifted to blood position. Pick cell to shift at␣
↪random,
    # but check it isn't the end_point cell.
    if i_dup > n_is - 2 or i_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, i_dup {i_dup}")
        out_of_body = True
    if j_dup > n_js - 2 or j_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, j_dup {j_dup}")
```

```
        out_of_body = True
    if k_dup > n_ks - 2 or k_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, k_dup {k_dup}")
        out_of_body = True
    if out_of_body:
        return  body_steer, neighbour, end_point, out_of_body
    #
    # Look first if can find a neighbour which is not a tumour cell
    n = 0
    neighbours = np.zeros((6, 3)).astype(int)
    n_to_e_sq = np.zeros(6)
    n_test = i_dup - 1, j_dup, k_dup
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    n_test = i_dup + 1, j_dup, k_dup
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    n_test = i_dup, j_dup - 1, k_dup
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    n_test = i_dup, j_dup + 1, k_dup
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    n_test = i_dup, j_dup, k_dup - 1
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    n_test = i_dup, j_dup, k_dup + 1
    if body[n_test] != c_tumour and not np.all(n_test == end_point):
        neighbours[n, :] = n_test
        n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
        n += 1
    #
    # No neighbour found yet. Allow it to be a tumour cell and try again.
    if n < 1:
        n_test = i_dup - 1, j_dup, k_dup
        if not np.all(n_test == end_point):
```

```
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        n_test = i_dup + 1, j_dup, k_dup
        if not np.all(n_test == end_point):
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        n_test = i_dup, j_dup - 1, k_dup
        if not np.all(n_test == end_point):
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        n_test = i_dup, j_dup + 1, k_dup
        if not np.all(n_test == end_point):
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        n_test = i_dup, j_dup, k_dup - 1
        if not np.all(n_test == end_point):
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        n_test = i_dup, j_dup, k_dup + 1
        if not np.all(n_test == end_point):
            neighbours[n, :] = n_test
            n_to_e_sq[n] = np.sum((neighbours[n, :] - end_point[:])**2)
            n += 1
        #
    neighbours = neighbours[0:n]
    n_to_e_sq = n_to_e_sq[0:n]
    #
    if n < 1:
        print(" ")
        print("No neighbour found - stop")
        sys.exit(0)
    #
    i_n_to_e_sort = np.argsort(n_to_e_sq)
    neighbour = neighbours[i_n_to_e_sort[0]]
    if debug:
        print(" ")
        print("Number in n_to_e_sq", n)
        print("neighbours \n",neighbours)
        print("n_to_e_sq)", n_to_e_sq)
        print("i_n_to_e_sort", i_n_to_e_sort)
        print("i_n_to_e_sort[0]", i_n_to_e_sort[0])
        print("neighbour", neighbour)
```

```
    #
    body_steer[neighbour[0], neighbour[1], neighbour[2]] = c_neighbour
    if debug:
        print(" ")
        print(f"duplicate {duplicate}")
        print(f"end_point {end_point}")
        print(f"neighbour {neighbour}")
    if with_plots:
        #
        # Make plot showing neighbour, cell to be duplicated and end cell
        print(" ")
        print("duplicate", i_dup, j_dup, k_dup)
        print("number of duplicates", np.sum(body_steer == c_duplicate))
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
        plot_dup = True
        plot_ext = True
        plot_end = True
        plot_neigh = True
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"Neighbour ({neighbour[0]}, {neighbour[1]}, {neighbour[2]})"
        plot_body(title, body_steer, body_steer, test_data, *neighbour,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    return body_steer, neighbour, end_point, out_of_body
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:41.592934

Date and time 2026-02-04 10:29:41.594555
Time since last check is 0:00:00.001621
```

Slack position finder Return to ToC: »

[12]:
```python
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def position_finder_slack(body_steer, i_dup, j_dup, k_dup, with_plots):
    '''
    Find the "neighbour" cell position to accommodate the "duplicate" and the "end" position
    to accommodate the neighbour. The end is a position outside the prostate and tumour.
    In this version, any possible end cell is used, not necessarily that closest to the duplicate.
    An attempt is first made to find a neighbour that is not part of the tumour
    and after that, tumour cells are allowed to be neighbours.
    '''
    debug = False
    #
    neighbour = np.zeros(3).astype(int)
    end_point = np.zeros(3).astype(int)
    #
    # Check whether there are too many duplicates (can only do one at a time!)
    n_duplicates = np.sum(body_steer == c_duplicate)
    if n_duplicates > 1:
        print(" ")
        print(f"Found {n_duplicates} duplicates in position_finder - stop")
        sys.exit(0)
    #
    # Find distances from cell to be duplicated to exterior cells
    # If find_exterior has already been run, don't do it again!
    n_ext = np.sum(body_steer == c_exterior)
    if n_ext < 1:
        body_steer, out_of_body = find_exterior(with_plots)
        if out_of_body:
            return body_steer, neighbour, end_point, out_of_body
    #
    n_ext = np.sum(body_steer == c_exterior)
    d_to_e_sq = np.zeros(n_ext)
    i_d_to_e, j_d_to_e, k_d_to_e = np.where(body_steer == c_exterior)
    d_to_e_sq = (i_dup - i_d_to_e)**2 + (j_dup - j_d_to_e)**2 + (k_dup - k_d_to_e)**2
```

```
    #
    # Find indices in dup_to_ext_sq that give minimum distance
    min_dist_ind = np.argsort(d_to_e_sq)
    #
    # Randomly order min_dist_ind where there are groups of distances that are␣
↪the same
    # (Not sure that this is necessary!)
    new_dist_ind = np.zeros(n_ext).astype(int)
    #
    un_dist_array, un_dist_ind, n_un_dist = np.unique(d_to_e_sq, return_index =␣
↪True,
                                                      return_inverse = False,
                                                      return_counts = True)
    n_groups = len(n_un_dist)
    n_bot = 0
    for n in range(0, n_groups):
        n_top = n_bot + n_un_dist[n]
        inds_here = np.zeros(n_un_dist[n]).astype(int)
        inds_here[0:n_un_dist[n]] = min_dist_ind[n_bot:n_top]
        rng.shuffle(inds_here)
        new_dist_ind[n_bot:n_top] = inds_here[0:n_un_dist[n]]
        n_bot = n_top
    #
    # Use below rather than above if don't want to do explicit random ordering!
    #new_dist_ind = min_dist_ind
    #
    # Define "end" cell, checking that it is not same as duplicate.
    duplicate = np.array([i_dup, j_dup, k_dup]).astype(int)
    origin = np.zeros(3).astype(int)
    end_test = np.zeros(3).astype(int)
    if debug:
        print(" ")
        print("Find end point")
        print("No. in exterior, len(min_dist_ind)",len(min_dist_ind))
        print("i_dup, j_dup, k_dup",i_dup, j_dup, k_dup)
    #
    w = 0
    out_of_body = False
    while w < len(new_dist_ind) and np.all(end_point == origin):
        min_ind = new_dist_ind[w]
        end_test[0] = i_d_to_e[min_ind]
        end_test[1] = j_d_to_e[min_ind]
        end_test[2] = k_d_to_e[min_ind]
        if debug:
            print(f"w {w}, end_point {end_point}, end_test {end_test}")
        if not np.all(end_test == duplicate):
            min_d_to_e = min_ind
```

```
            end_point[:] = end_test[:]
        w += 1
    if np.all(end_point == origin):
        print(" ")
        print("No end point found - stop")
        sys.exit(0)
    if debug:
        print(" ")
        print(f"duplicate {duplicate}")
        print(f"end_point {end_point}")
    #
    i_end, j_end, k_end = end_point[0], end_point[1], end_point[2]
    if i_end > n_is - 2 or i_end < 2:
        print(" ")
        print(f"End point out of body, i_end {i_end}")
        out_of_body = True
    if j_end > n_js - 2 or j_end < 2:
        print(" ")
        print(f"End point out of body, j_end {j_end}")
        out_of_body = True
    if k_end > n_ks - 2 or k_end < 2:
        print(" ")
        print(f"End point out of body, k_end {k_end}")
        out_of_body = True
    if out_of_body:
        return body_steer, neighbour, end_point, out_of_body
    #
    body_steer[i_end, j_end, k_end] = c_end
    #
    if with_plots:
        #
        # Make plot showing the cell to be duplicated and end cell
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
        plot_dup = True
        plot_ext = True
        plot_end = True
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
```

```
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"End ({i_end}, {j_end}, {k_end})"
        plot_body(title, body_steer, body_steer, test_data, i_end, j_end, k_end,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    # Choose "neighbour" cell to shift to end position. Cell adjacent to␣
↪duplicate can be prostate,
    # tumour or blood. Will be shifted to blood position. Pick cell to shift at␣
↪random,
    # but check it isn't the end_point cell.
    if i_dup > n_is - 2 or i_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, i_dup {i_dup}")
        out_of_body = True
    if j_dup > n_js - 2 or j_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, j_dup {j_dup}")
        out_of_body = True
    if k_dup > n_ks - 2 or k_dup < 2:
        print(" ")
        print(f"Duplicate point out of body, k_dup {k_dup}")
        out_of_body = True
    if out_of_body:
        return  body_steer, neighbour, end_point, out_of_body
    #
    chooser = np.linspace(0, 5, 6).astype(int)
    rng.shuffle(chooser)
    neighbours = np.zeros((6, 3)).astype(int)
    neighbours[0, :] = np.array([i_dup - 1, j_dup, k_dup])
    neighbours[1, :] = np.array([i_dup + 1, j_dup, k_dup])
    neighbours[2, :] = np.array([i_dup, j_dup - 1, k_dup])
    neighbours[3, :] = np.array([i_dup, j_dup + 1, k_dup])
    neighbours[4, :] = np.array([i_dup, j_dup, k_dup - 1])
    neighbours[5, :] = np.array([i_dup, j_dup, k_dup + 1])
    #
    origin = np.zeros(3).astype(int)
    if debug:
        print(" ")
        print("Find neighbour")
    #
```

```
    # Find neighbour that is not tumour cell
    w = 0
    while w < 6 and np.all(neighbour == origin):
        if not np.logical_or(np.all(neighbours[chooser[w], :] == end_point),
                                            body[*neighbours[chooser[w]]] ==␣
↪c_tumour):
            neighbour[:] = neighbours[w, :]
        if debug:
            print(f"neighbour one, w {w}, end_point {end_point}, end_test␣
↪{end_test}")
        w += 1
    #
    # If no neighbour found, look again allowing it to be a tumour cell
    if np.all(neighbour == origin):
        w = 0
        while w < 6 and np.all(neighbour == origin):
            if not np.all(neighbours[chooser[w], :] == end_point):
                neighbour[:] = neighbours[chooser[w], :]
            if debug:
                print(f"neighbour two, w {w}, end_point {end_point}, end_test␣
↪{end_test}")
            w += 1
        #
    if np.all(neighbour == origin):
        print(" ")
        print("No neighbour found - stop")
        sys.exit(0)
    #
    body_steer[neighbour[0], neighbour[1], neighbour[2]] = c_neighbour
    if debug:
        print(" ")
        print(f"duplicate {duplicate}")
        print(f"end_point {end_point}")
        print(f"neighbour {neighbour}")
    if with_plots:
        #
        # Make plot showing neighbour, cell to be duplicated and end cell
        print(" ")
        print("duplicate", i_dup, j_dup, k_dup)
        print("number of duplicates", np.sum(body_steer == c_duplicate))
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = False
        plot_prostate = False
        plot_tumour = False
        plot_surface = True
```

```
        plot_dup = True
        plot_ext = True
        plot_end = True
        plot_neigh = True
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = f"Neighbour ({neighbour[0]}, {neighbour[1]}, {neighbour[2]})"
        plot_body(title, body_steer, body_steer, test_data, *neighbour,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    return body_steer, neighbour, end_point, out_of_body
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:42.274129

Date and time 2026-02-04 10:29:42.275373
Time since last check is 0:00:00.001244
```

1.5.12 Shift cells to new positions

Return to ToC: »

[13]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def cell_shifter(duplicate, neighbour, end_point, with_plots):
    '''
    Shift the "neighbour" cell in the body array to the "end" position and the
    "duplicate" to the neighbour position.
    '''
    debug = False
    #
    if debug:
```

```
        print(f" ")
        print(f"duplicate {duplicate}")
        print(f"end_point {end_point}")
        print(f"neighbour {neighbour}")
        print(" ")
        print("Initial values")
        print("body[*duplicate]",body[*duplicate])
        print("body[*neighbour]",body[*neighbour])
        print("body[*end_point]",body[*end_point])
    #
    body[*end_point] = body[*neighbour]
    body[*neighbour] = body[*duplicate]
    body[*duplicate] = c_tumour
    #
    if debug:
        print(" ")
        print("New values")
        print("body[*duplicate]",body[*duplicate])
        print("body[*neighbour]",body[*neighbour])
        print("body[*end_point]",body[*end_point])
    #
    if with_plots:
        #
        # Make plot showing results of copying
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = True
        plot_prostate = True
        plot_tumour = True
        plot_surface = False
        plot_dup = False
        plot_ext = False
        plot_end = False
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = "After PSA update"
        plot_body(title, body, PSA, test_data, i_end, j_end, k_end,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
```

```
                    plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                    plot_2D, plot_log, show_legends,
                    c_map, v_min = v_min, v_max = v_max)
        #
    return body
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:42.950970

Date and time 2026-02-04 10:29:42.951815
Time since last check is 0:00:00.000845
```

1.5.13 Update PSA array

Return to ToC: »

[14]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def PSA_updater(PSA, duplicate, neighbour, endpoint, with_plots):
    '''
    Update the PSA array using the "neighbour", "end", and "duplicate"␣
 ↪positions.
    '''
    debug = False
    #
    PSA[*end_point] = PSA[*neighbour]
    PSA[*neighbour] = PSA[*duplicate]
    PSA[*duplicate] = np.mean(PSA[body == c_tumour])
    #
    if with_plots:
        #
        # Make plot showing results of updating
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = True
        plot_prostate = True
        plot_tumour = True
        plot_surface = False
        plot_dup = False
        plot_ext = False
        plot_end = False
```

```
        plot_neigh = False
        plot_test = False
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        #c_map = cm.viridis
        c_map = cm.plasma
        #
        title = "After PSA update"
        plot_body(title, body, PSA, test_data, i_end, j_end, k_end,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
 ↪plot_surface,
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
        #
    return PSA
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:43.542842

Date and time 2026-02-04 10:29:43.543390
Time since last check is 0:00:00.000548
```

1.5.14 Reset body steering values

Return to ToC: »

[15]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
def body_steer_reset(with_plots):
    '''
    Reset indices in body_steer used to indicate surface cells, cell to␣
 ↪duplicate,
    exterior cells etc.
    '''
    #
    # Surface
    bool_sel = np.where(body_steer == c_surface)
    body_steer[bool_sel] = 0
```

```
    #
    # Duplicate
    bool_sel = np.where(body_steer == c_duplicate)
    body_steer[bool_sel] = 0
    #
    # Exterior
    bool_sel = np.where(body_steer == c_exterior)
    body_steer[bool_sel] = 0
    #
    # End
    bool_sel = np.where(body_steer == c_end)
    body_steer[bool_sel] = 0
    #
    # Neighbour
    bool_sel = np.where(body_steer == c_neighbour)
    body_steer[bool_sel] = 0
    #
    # Test
    bool_sel = np.where(body_steer == c_test)
    body_steer[bool_sel] = 0
    #
    if with_plots:
        #
        # Make plot showing result of reset
        v_min = -1.0
        v_max = -1.0
        plot_point = False
        plot_blood = True
        plot_prostate = True
        plot_tumour = True
        plot_surface = True
        plot_dup = True
        plot_ext = True
        plot_end = True
        plot_neigh = True
        plot_test = True
        #
        plot_2D = True
        plot_log = False
        show_legends = True
        #
        c_map = cm.viridis
        #
        title = "After reset"
        plot_body(title, body_steer, body_steer, test_data, i_dup, j_dup, k_dup,
                  plot_point, plot_blood, plot_prostate, plot_tumour,␣
↪plot_surface,
```

```
                  plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
                  plot_2D, plot_log, show_legends,
                  c_map, v_min = v_min, v_max = v_max)
    #
    return body_steer
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:29:44.149902

Date and time 2026-02-04 10:29:44.150499
Time since last check is 0:00:00.000597
```

1.6 Overview of running of model

In order to investigate changes in PSA as prostate cancer develops: 1) Initialise body structure: border, blood, and prostate. 2) Initialise PSA levels. 3) Run diffusion to get a "zero" point. 4) Add a tumour cell or cells. 5) Initialise PSA levels with the tumour. 6) Run diffusion. 7) Grow the tumour. 8) Run diffusion.

Repeat 7) and 8) until maximum required (or possible) tumour size reached.
After each run of diffusion, record the PSA level in the blood and other relevant variables.

Return to ToC: »

1.7 Run complete model

Return to ToC: »

[16]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
# Set number of PSA arrays
# triple True and double True implies use PSA_top, PSA_mid and PSA_bot
# triple False and double True implies use PSA_top and PSA_bot
# triple and double False implies use PSA_top only.
# To get top and bottom limits (using triple or double), set with_init True.
# If have wuth_init True and triple and double False, PSA_top array will be␣
 ↪reinitialised
# after each duplication. If with_init is False, initialisation only happens␣
 ↪when the
# body is first created.
triple = False
double = False
with_init = False
```

```
growth_rate = 0.2
grow_near = True
#n_real_times = 5
n_real_times = 25
#n_real_times = 45
#
# Set cell codes
#
c_test = -1
c_blood = 1
c_prostate = 2
c_tumour = 3
c_next = 4
#
c_surface = 1
c_duplicate = 2
c_exterior = 3
c_end = 4
c_neighbour = 5
#
# Initialise body without tumour
#
debug = False
with_plots = False
#
# Number of cells
#n_cells_x, n_cells_y, n_cells_z = 25, 27, 29
n_cells_x, n_cells_y, n_cells_z = 45, 47, 49
#n_cells_x, n_cells_y, n_cells_z = 50, 52, 54
#n_cells_x, n_cells_y, n_cells_z = 60, 62, 64
#n_cells_x, n_cells_y, n_cells_z = 70, 72, 74
#n_cells_x, n_cells_y, n_cells_z = 84, 84, 84
#
# Number of cells in body array
n_is = n_cells_x
n_js = n_cells_y
n_ks = n_cells_z
n_body = n_is*n_js*n_ks
body = np.zeros((n_is, n_js, n_ks)).astype(int)
body_steer = np.zeros((n_is, n_js, n_ks)).astype(int)
test_data = np.zeros((n_is, n_js, n_ks))
#
print(" ")
print(f"Body dimensions {n_is} in x, {n_js} in y, {n_ks} in z.")
print(f"Body size {n_body} cells.")
#
# Define prostate centre and radius
```

```
prostate_rad = 10
prostate_rad_2 = prostate_rad**2
#
# Central prostate
i_prostate = n_cells_x//2
j_prostate = n_cells_y//2
k_prostate = n_cells_z//2
#
# Off-centre prostate
#i_prostate = int(2*prostate_rad)
#j_prostate = int(2*prostate_rad)
#k_prostate = int(2*prostate_rad)
#
# Define tumour centre and radius (radius between 0.5 and 1 to get single␣
 ↪initial cell!)
tumour_rad = 0.6
tumour_rad_2 = tumour_rad**2
#
# Central tumour
#i_tumour = i_prostate
#j_tumour = j_prostate
#k_tumour = k_prostate
#
# Nearly central tumour
#i_tumour = i_prostate - prostate_rad//3
#j_tumour = j_prostate - prostate_rad//4
#k_tumour = k_prostate + prostate_rad//4
#
# Internal tumour, deepish but off centre
i_tumour = i_prostate + prostate_rad//2 - 1
j_tumour = j_prostate + prostate_rad//2 - 1
k_tumour = k_prostate + prostate_rad//2 - 1
#
# Internal tumour close to surface
#i_tumour = i_prostate + prostate_rad//2
#j_tumour = j_prostate + prostate_rad//2
#k_tumour = k_prostate + prostate_rad//2
#
# Peripheral tumour
#i_tumour = i_prostate + int(prostate_rad/np.sqrt(2)) - 1
#j_tumour = j_prostate + int(prostate_rad/np.sqrt(2)) - 1
#k_tumour = k_prostate + int(prostate_rad/np.sqrt(2)) - 1
#
print(" ")
print(f"Prostate centre, indices {i_prostate}, {j_prostate}, {k_prostate}")
print(f"Prostate radius {prostate_rad}")
print(f"Tumour centre, indices {i_tumour}, {j_tumour}, {k_tumour}")
```

```
print(f"Tumour radius {tumour_rad}")
#
index = 0
for i in range(0, n_is):
    for j in range(0, n_js):
        for k in range(0, n_ks):
            dist_prostate_2 = ((i - i_prostate)**2 +
                               (j - j_prostate)**2 +
                               (k - k_prostate)**2)
            #
            if dist_prostate_2 < prostate_rad_2:
                body[i, j, k] = c_prostate
                #
            else:
                body[i, j, k] = c_blood
            #
        #
    #
n_blood_init = np.sum(body == c_blood)
n_prostate_init = np.sum(body == c_prostate)
n_tumour_init = np.sum(body == c_tumour)
print(" ")
print(f"Initial number of blood cells is {n_blood_init}")
print(f"Initial number of cells in prostate is {n_prostate_init}")
print(f"Initial number of cells in tumour is {n_tumour_init}")
print(f"Tumour growth rate is {growth_rate:.3f}")
if grow_near:
    print(f"Near tumour growth used.")
else:
    print(f"Slack tumour growth used.")
#
if with_plots:
    plot_point = False
    plot_blood = True
    plot_prostate = True
    plot_tumour = True
    plot_surface = False
    plot_dup = False
    plot_ext = False
    plot_end = False
    plot_neigh = False
    plot_test = False
    #
    plot_2D = True
    plot_log = False
    show_legends = True
    #
```

```
    c_map = cm.viridis
    #c_map = cm.plasma
    #
    v_min = -1.0
    v_max = -1.0
    plot_body("Body", body, body_steer, test_data, i_tumour, j_tumour, k_tumour,
              plot_point, plot_blood, plot_prostate, plot_tumour, plot_surface,
              plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
              plot_2D, plot_log, show_legends,
              c_map, v_min = v_min, v_max = v_max)
#
# Initialise PSA without tumour
# PSA decay rates set 1
#tau_prostate = 1/20.0
#tau_tumour = tau_prostate
#tau_blood = 1/2.0
#
# PSA production rates set
#sig_prostate = 2.0
#sig_tumour = 20.0
#sig_blood = 0.0
#
# PSA decay rates set 2
tau_prostate = 0.000217
tau_tumour = tau_prostate
tau_blood = 0.000144
#
# PSA production rates set 2
sig_prostate = 0.00802
sig_tumour = 0.96
sig_blood = 0.0
#
# Blood dilution factor
dil_blood = 0.02
#
# Set "diffusion" coefficient
beta = 0.1
#
# Allow for a proportion of random variation if required
r_prop = 0.0
#
# Set up large scale time array
times = np.linspace(0, n_real_times - 1, n_real_times).astype(int)
i_time = 0
print(f"--------------------------------------------------------------------------------",
      f"\b------------------------")
print(f"Time {times[i_time]}")
```

```
#
print("Initialise PSA_top, no tumour")
set_level = 2
PSA_top = initialise_PSA(set_level, r_prop, with_plots, i_prostate, j_prostate,␣
 ↪k_prostate)
if triple:
    print("Initialise PSA_mid, no tumour")
    set_level = 1
    PSA_mid = initialise_PSA(set_level, r_prop, with_plots, i_prostate,␣
 ↪j_prostate, k_prostate)
if double:
    print("Initialise PSA_bot, no tumour")
    set_level = 0
    PSA_bot = initialise_PSA(set_level, r_prop, with_plots, i_prostate,␣
 ↪j_prostate, k_prostate)
#
# Simulate diffusion, creation and decay of PSA without tumour
#
n_times = 250
conv_test = 0.001
with_all_plots = False
print("Diffusion PSA_top, no tumour")
set_level = 2
PSA_top = do_diffusion(PSA_top, beta, r_prop, n_times, conv_test,
                       with_init, set_level, with_all_plots,
                       i_prostate, j_prostate, k_prostate)
if triple:
    print("Diffusion PSA_mid, no tumour")
    set_level = 1
    with_init_here = False
    PSA_mid = do_diffusion(PSA_mid, beta, r_prop, n_times, conv_test,
                           with_init_here, set_level, with_all_plots,
                           i_prostate, j_prostate, k_prostate)
if double:
    print("Diffusion PSA_bot, no tumour")
    set_level = 0
    PSA_bot = do_diffusion(PSA_bot, beta, r_prop, n_times, conv_test,
                           with_init, set_level, with_all_plots,
                           i_prostate, j_prostate, k_prostate)
#
# Store PSA level at this time
#
PSA_top_time = np.zeros(n_real_times)
PSA_mid_time = np.zeros(n_real_times)
PSA_bot_time = np.zeros(n_real_times)
#
area_time = np.zeros(n_real_times)
```

```
btcell_time = np.zeros(n_real_times)
tumour_time = np.zeros(n_real_times)
prostate_time = np.zeros(n_real_times)
blood_time = np.zeros(n_real_times)
#
PSA_top_level = np.mean(PSA_top[body == c_blood])
if triple:
    PSA_mid_level = np.mean(PSA_mid[body == c_blood])
if double:
    PSA_bot_level = np.mean(PSA_bot[body == c_blood])
#
tumour_cells = np.sum(body == c_tumour)
prostate_cells = np.sum(body == c_prostate)
blood_cells = np.sum(body == c_blood)
#
PSA_top_time[i_time] = PSA_top_level
if triple:
    PSA_mid_time[i_time] = PSA_mid_level
if double:
    PSA_bot_time[i_time] = PSA_bot_level
#
tumour_time[i_time] = tumour_cells
prostate_time[i_time] = prostate_cells
blood_time[i_time] = blood_cells
print(f"At time {times[i_time]}:")
print(f"No. tumour cells {tumour_cells}, "
      f"prostate cells {prostate_cells}, blood cells {blood_cells}")
print(f"PSA_top_level {PSA_top_level:.6e}")
if triple:
    print(f"PSA_mid_level {PSA_mid_level:.6e}")
if double:
    print(f"PSA_bot_level {PSA_bot_level:.6e}")
#
# Add initial tumour
#
i_time = 1
print(f"-------------------------------------------------------------------------------",
      f"\b------------------------")
print(f"Time {times[i_time]}")
#
debug = False
with_plots = True
#
index = 0
for i in range(0, n_is):
    for j in range(0, n_js):
        for k in range(0, n_ks):
```

```
            dist_tumour_2 = ((i - i_tumour)**2 +
                             (j - j_tumour)**2 +
                             (k - k_tumour)**2)
            #
            if dist_tumour_2 < tumour_rad_2:
                body[i, j, k] = c_tumour
            #
        #
    #
n_blood_init = np.sum(body == c_blood)
n_prostate_init = np.sum(body == c_prostate)
n_tumour_init = np.sum(body == c_tumour)
print(" ")
print(f"Number of blood cells is {n_blood_init}")
print(f"Number of cells in prostate is {n_prostate_init}")
print(f"Number of cells in tumour is {n_tumour_init}")
#
if with_plots:
    plot_point = False
    plot_blood = True
    plot_prostate = True
    plot_tumour = True
    plot_surface = False
    plot_dup = False
    plot_ext = False
    plot_end = False
    plot_neigh = False
    plot_test = False
    #
    plot_2D = True
    plot_log = False
    show_legends = True
    #
    c_map = cm.viridis
    #c_map = cm.plasma
    #
    v_min = -1.0
    v_max = -1.0
    plot_body("Body", body, body_steer, test_data, i_tumour, j_tumour, k_tumour,
              plot_point, plot_blood, plot_prostate, plot_tumour, plot_surface,
              plot_dup, plot_ext, plot_end, plot_neigh, plot_test,
              plot_2D, plot_log, show_legends,
              c_map, v_min = v_min, v_max = v_max)
#
# Simulate diffusion, creation and decay of PSA with initial tumour
#
print("Diffusion PSA_top, with initial tumour")
```

```
set_level = 2
PSA_top = do_diffusion(PSA_top, beta, r_prop, n_times, conv_test,
                       with_init, set_level, with_all_plots,
                       i_tumour, j_tumour, k_tumour)
if triple:
    print("Diffusion PSA_mid, with initial tumour")
    set_level = 1
    with_init_here = False
    PSA_mid = do_diffusion(PSA_mid, beta, r_prop, n_times, conv_test,
                           with_init_here, set_level, with_all_plots,
                           i_tumour, j_tumour, k_tumour)
if double:
    print("Diffusion PSA_bot, with initial tumour")
    set_level = 0
    PSA_bot = do_diffusion(PSA_bot, beta, r_prop, n_times, conv_test,
                           with_init, set_level, with_all_plots,
                           i_tumour, j_tumour, k_tumour)
#
# Store PSA level at this time
#
PSA_top_time[i_time] = PSA_top_level
if triple:
    PSA_mid_time[i_time] = PSA_mid_level
if double:
    PSA_bot_time[i_time] = PSA_bot_level
#
with_plots_here = True
i_plot, j_plot, k_plot = i_tumour, j_tumour, k_tumour
btcell, bt_area = area_tumour_blood(with_plots_here, i_time, i_plot, j_plot,␣
 ↪k_plot)
#
area_time[i_time] = bt_area
btcell_time[i_time] = btcell
tumour_time[i_time] = tumour_cells
prostate_time[i_time] = prostate_cells
blood_time[i_time] = blood_cells
print(f"At time {times[i_time]}:")
print(f"No. tumour cells {tumour_cells}, "
      f"prostate cells {prostate_cells}, blood cells {blood_cells}")
print(f"PSA_top_level {PSA_top_level:.6e}")
if triple:
    print(f"PSA_mid_level {PSA_mid_level:.6e}")
if double:
    print(f"PSA_bot_level {PSA_bot_level:.6e}")
#
i_time = 2
out_of_body = False
```

```
while i_time < n_real_times and not out_of_body:
    #
    # Increase tumour size - make duplications
    #
    ␣
 ↪print(f"-------------------------------------------------------------------------------------
          f"\b-----------------------")
    print(f"Time {times[i_time]}")
    body_org = np.zeros((n_is, n_js, n_ks))
    body_org[:] = body[:]
    #
    n_duplicates = int(np.exp(growth_rate*(i_time - 2)))
    #
    with_plots = False
    for i in range(0, n_duplicates):
        duplicate, body_steer, out_of_body = find_duplicator(with_plots)
        if out_of_body:
            break
        #
        if grow_near:
            body_steer, neighbour, end_point, out_of_body = \
                position_finder_near(body_steer, *duplicate, with_plots)
        else:
            body_steer, neighbour, end_point, out_of_body = \
                position_finder_slack(body_steer, *duplicate, with_plots)
        if out_of_body:
            break
        #
        body = cell_shifter(duplicate, neighbour, end_point, with_plots)
        PSA_top = PSA_updater(PSA_top, duplicate, neighbour, end_point,␣
 ↪with_plots)
        if triple:
            PSA_mid = PSA_updater(PSA_mid, duplicate, neighbour, end_point,␣
 ↪with_plots)
        if double:
            PSA_bot = PSA_updater(PSA_bot, duplicate, neighbour, end_point,␣
 ↪with_plots)
            #
        body_steer = body_steer_reset(with_plots)
    #
    if out_of_body:
        break
    with_plots = False
    title = "$\\delta$ " + str(n_duplicates) + " duplications"
    show_change(body_org, i_tumour, j_tumour, k_tumour, with_plots, title)
    #
    # Simulate diffusion, creation and decay of PSA with enlarged tumour
```

```
    print("Diffusion PSA_top, with tumour")
    set_level = 2
    PSA_top = do_diffusion(PSA_top, beta, r_prop, n_times, conv_test,
                           with_init, set_level, with_all_plots,
                           i_tumour, j_tumour, k_tumour)
    if triple:
        print("Diffusion PSA_mid, with tumour")
        set_level = 1
        with_init_here = False
        PSA_mid = do_diffusion(PSA_mid, beta, r_prop, n_times, conv_test,
                               with_init_here, set_level, with_all_plots,
                               i_tumour, j_tumour, k_tumour)
    if double:
        print("Diffusion PSA_bot, with tumour")
        set_level = 0
        PSA_bot = do_diffusion(PSA_bot, beta, r_prop, n_times, conv_test,
                               with_init, set_level, with_all_plots,
                               i_tumour, j_tumour, k_tumour)
    #
    # Store PSA level at this time
    PSA_top_level = np.mean(PSA_top[body == c_blood])
    if triple:
        PSA_mid_level = np.mean(PSA_mid[body == c_blood])
    if double:
        PSA_bot_level = np.mean(PSA_bot[body == c_blood])
    #
    tumour_cells = np.sum(body == c_tumour)
    prostate_cells = np.sum(body == c_prostate)
    blood_cells = np.sum(body == c_blood)
    #
    PSA_top_time[i_time] = PSA_top_level
    if triple:
        PSA_mid_time[i_time] = PSA_mid_level
    if double:
        PSA_bot_time[i_time] = PSA_bot_level
    #
    # Determine area of tumour in direct contact with blood
    with_plots_here = True
    i_plot, j_plot, k_plot = i_tumour, j_tumour, k_tumour
    btcell, bt_area = area_tumour_blood(with_plots_here, i_time, i_plot,␣
↪j_plot, k_plot)
    btcell_time[i_time] = btcell
    area_time[i_time] = bt_area
    tumour_time[i_time] = tumour_cells
    prostate_time[i_time] = prostate_cells
    blood_time[i_time] = blood_cells
    print(f"At time {times[i_time]}:")
```

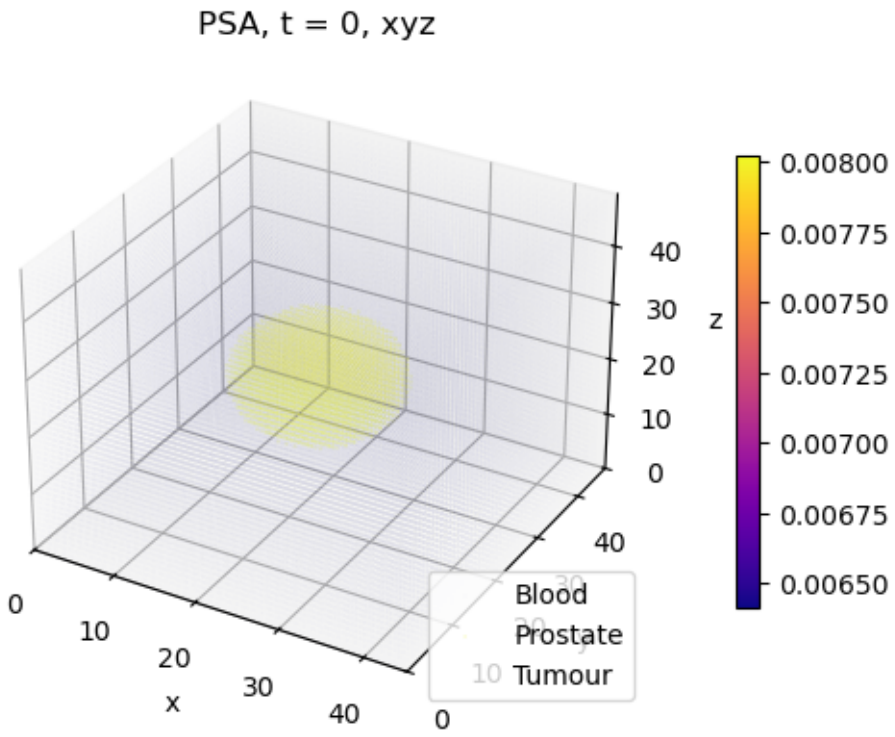
```
    print(f"No. tumour cells {tumour_cells}, "
          f"prostate cells {prostate_cells}, blood cells {blood_cells}")
    print(f"PSA_top_level {PSA_top_level:.6e}")
    if triple:
        print(f"PSA_mid_level {PSA_mid_level:.6e}")
    if double:
        print(f"PSA_bot_level {PSA_bot_level:.6e}")
    i_time += 1
#
times = np.linspace(0, i_time - 1, i_time)
blood_time = blood_time[0:i_time]
prostate_time = prostate_time[0:i_time]
tumour_time = tumour_time[0:i_time]
area_time = area_time[0:i_time]
btcell_time = btcell_time[0:i_time]
#
PSA_top_time = PSA_top_time[0:i_time]
PSA_mid_time = PSA_mid_time[0:i_time]
PSA_bot_time = PSA_bot_time[0:i_time]
#
most_time = blood_time + prostate_time + tumour_time
PSA_cells = tumour_cells + prostate_cells
#
btcell_color = np.zeros(i_time).astype(str)
btcell_color[btcell_time > 0] = 'r'
btcell_color[btcell_time == 0] = 'b'
#
face_number = np.zeros(i_time)
face_number[tumour_time < 1] = 0.0
face_number[tumour_time > 0] = area_time[tumour_time > 0]/
 ↪tumour_time[tumour_time > 0]
#
fig = plt.figure(figsize = (6, 9))
#
ax = fig.add_subplot(3, 1, 1)
ax.set_title("Cells with time")
ax.set_xlabel("Time")
ax.set_ylabel("No. cells")
ax.plot(times, blood_time, marker = 'o', linestyle = ':', color = 'r', label =␣
 ↪"Blood")
ax.plot(times, tumour_time, marker = 'o', linestyle = ':', color = 'b', label =␣
 ↪"Tumour")
ax.plot(times, prostate_time, marker = 'o', linestyle = ':', color = 'c', label␣
 ↪= "Prostate")
ax.set_yscale('log')
ax.grid(color = 'g')
ax.legend()
```

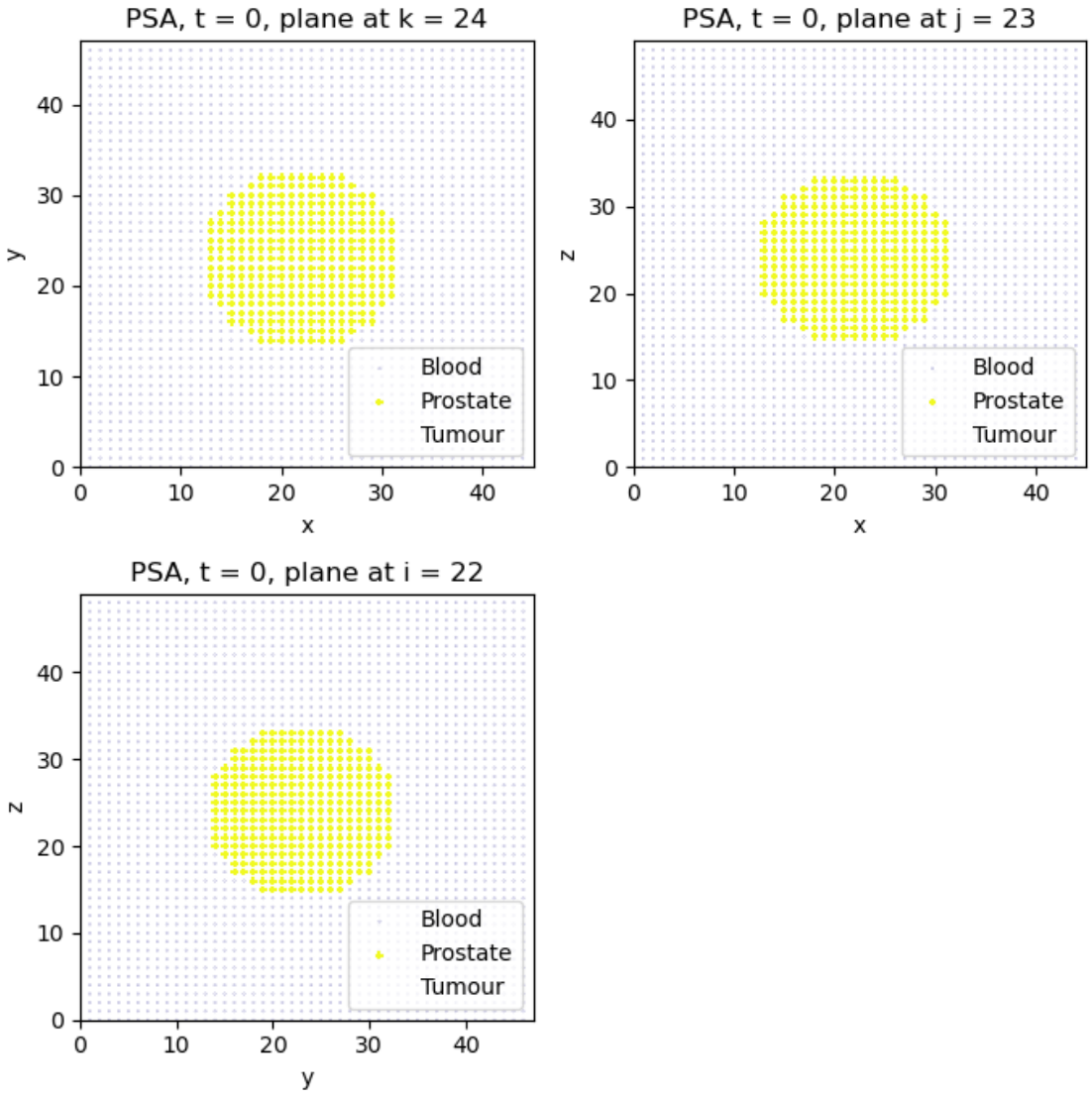
```
#
ax = fig.add_subplot(3, 1, 2)
ax.set_title("Tumour cells and interface area with time")
ax.set_xlabel("Time")
ax.set_ylabel("No. cells or cell faces")
ax.plot(times, tumour_time, marker = 'o', linestyle = ':', color = 'r', label =␣
 ↪"Tumour cells")
ax.plot(times, area_time, marker = 'o', linestyle = ':', color = 'b', label =␣
 ↪"Interface area")
ax.plot(times, face_number, marker = 'o', linestyle = ':', color = 'c',
        label = "Mean interface area per cell")
ax.set_yscale('log')
ax.grid(color = 'g')
ax.legend()
#
ax = fig.add_subplot(3, 1, 3)
ax.set_title("PSA level with time")
ax.set_xlabel("Time")
ax.set_ylabel("PSA")
if double:
    ax.plot(times, PSA_top_time, marker = 'v', linestyle = '', color = 'k')
    ax.plot(times, PSA_bot_time, marker = '^', linestyle = '', color = 'k')
    ax.fill_between(times, PSA_bot_time, PSA_top_time, color = 'k', alpha = 0.3)
    if triple:
        ax.plot(times, PSA_mid_time, marker = '+', linestyle = '-', color = 'k')
else:
    ax.plot(times, PSA_top_time, marker = 'o', linestyle = ':', color = 'k')
ax.grid(color = 'g')
#
plt.tight_layout()
plt.show()
#
PSA_cells_time = tumour_time + prostate_time
#
plot_log_here = False
#
fig = plt.figure(figsize = (6, 6))
#
ax = fig.add_subplot(2, 1, 1)
ax.set_title("PSA level as function of tumour cell number")
ax.set_xlabel("No. cells")
ax.set_ylabel("PSA")
if double:
    ax.scatter(tumour_time, PSA_top_time, marker = 'v', c = btcell_color)
    ax.scatter(tumour_time, PSA_bot_time, marker = '^', c = btcell_color)
    ax.fill_between(tumour_time, PSA_bot_time, PSA_top_time, color = 'c', alpha␣
 ↪= 0.3)
```

```
    if triple:
        ax.scatter(tumour_time, PSA_mid_time, marker = '+', c = btcell_color)
else:
    ax.scatter(tumour_time, PSA_top_time, marker = 'o', c = btcell_color)
    ax.plot(tumour_time, PSA_top_time, marker = '', linestyle = ':', color =␣
 ↪'c')
ax.grid(color = 'g')
if plot_log_here and len(tumour_time > 0):
    ax.set_xscale('log')
    ax.set_yscale('log')
#
ax = fig.add_subplot(2, 1, 2)
ax.set_title("PSA level as function of interface area")
ax.set_xlabel("Area")
ax.set_ylabel("PSA")
if double:
    ax.scatter(area_time, PSA_top_time, marker = 'v', c = btcell_color)
    ax.scatter(area_time, PSA_bot_time, marker = '^', c = btcell_color)
    ax.fill_between(area_time, PSA_bot_time, PSA_top_time, color = 'c', alpha =␣
 ↪0.3)
    if triple:
        ax.scatter(area_time, PSA_mid_time, marker = '+', c = btcell_color)
else:
    ax.scatter(area_time, PSA_top_time, marker = 'o', c = btcell_color)
    ax.plot(area_time, PSA_top_time, marker = '', linestyle = ':', color = 'c')
ax.grid(color = 'g')
if plot_log_here:
    ax.set_xscale('log')
    ax.set_yscale('log')
#
plt.tight_layout()
plt.show()
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

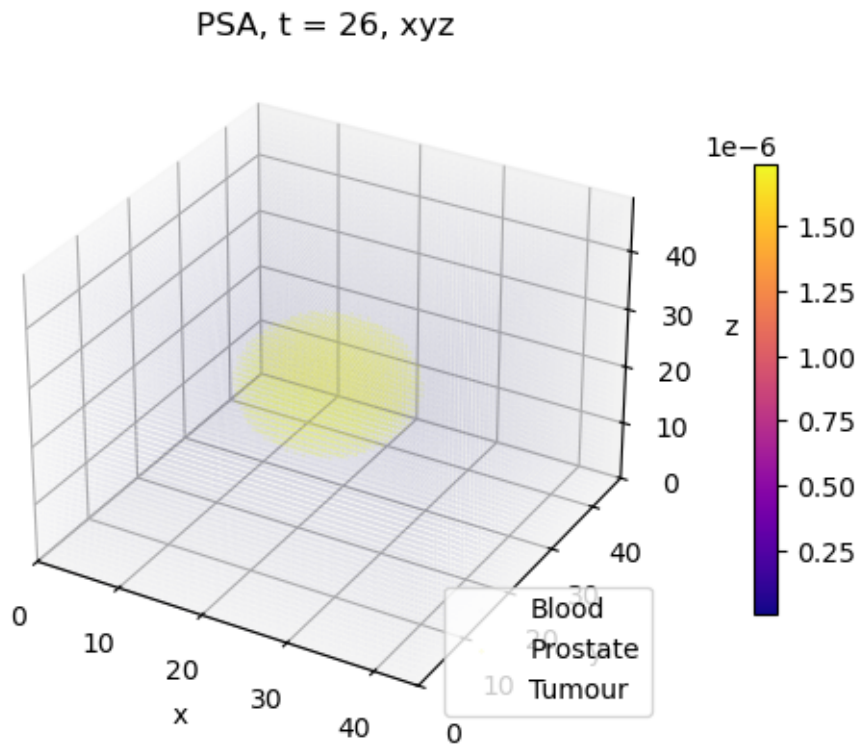
```
Date and time 2026-02-04 10:29:45.229726

Body dimensions 45 in x, 47 in y, 49 in z.
Body size 103635 cells.

Prostate centre, indices 22, 23, 24
Prostate radius 10
Tumour centre, indices 26, 27, 28
Tumour radius 0.6
```

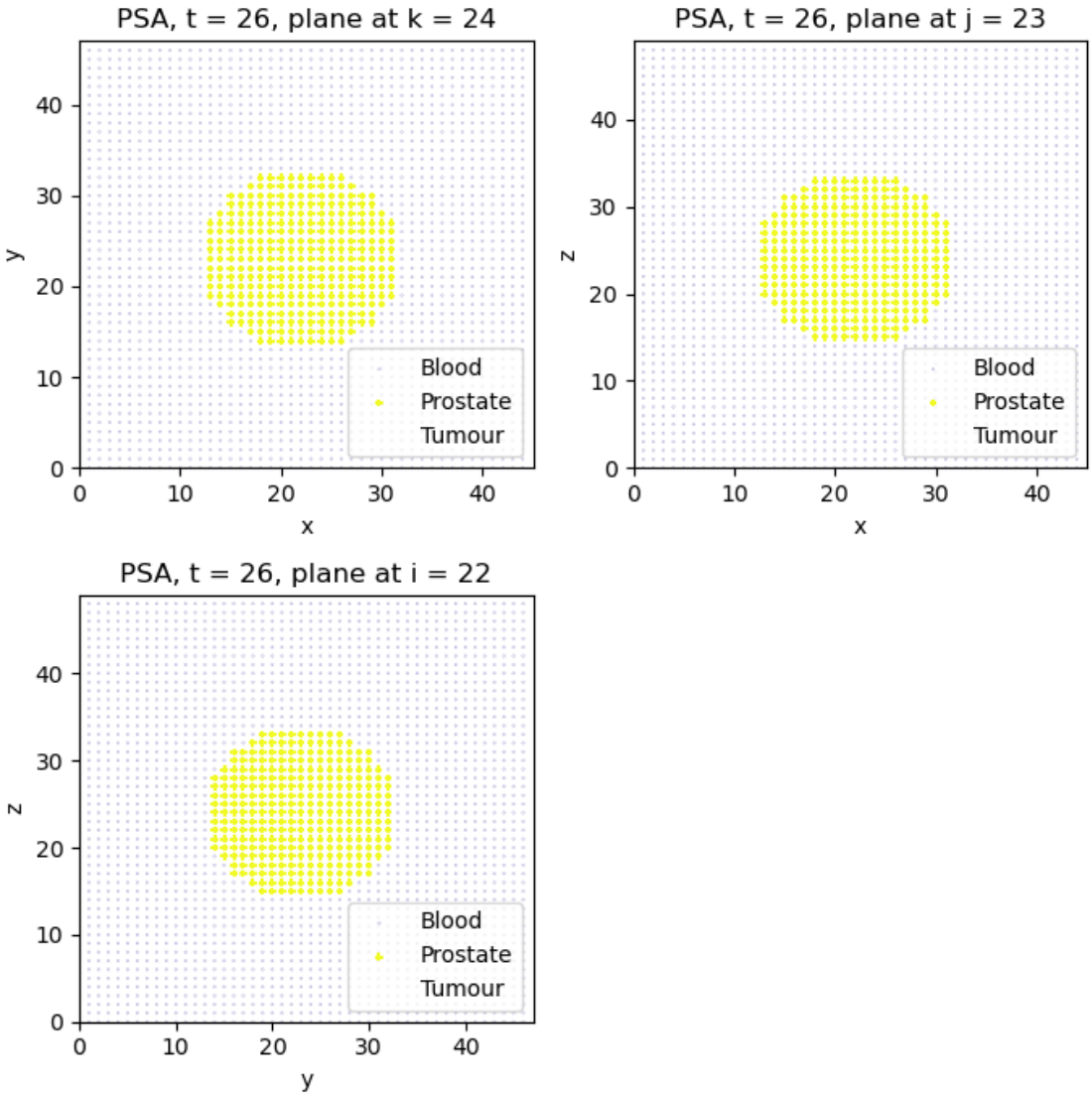
```
Initial number of blood cells is 99496
Initial number of cells in prostate is 4139
Initial number of cells in tumour is 0
Tumour growth rate is 0.200
Near tumour growth used.
--------------------------------------------------------------------------------
--------------------------
Time 0
Initialise PSA_top, no tumour
```

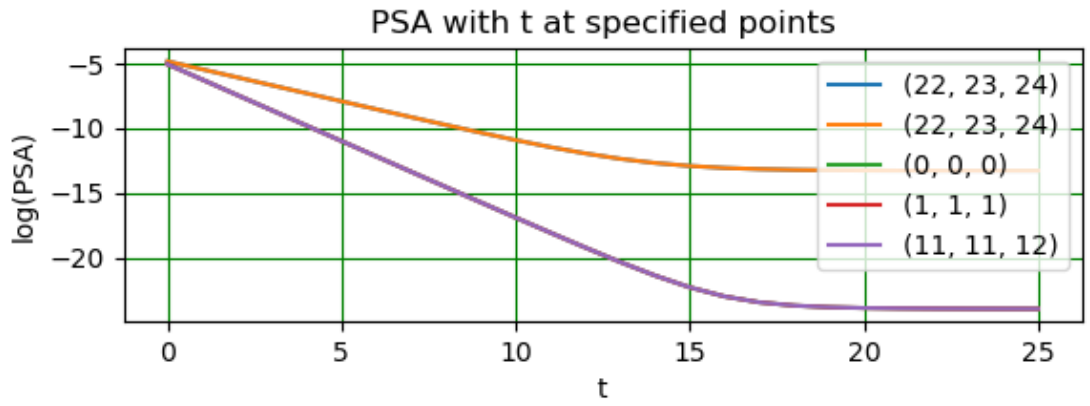


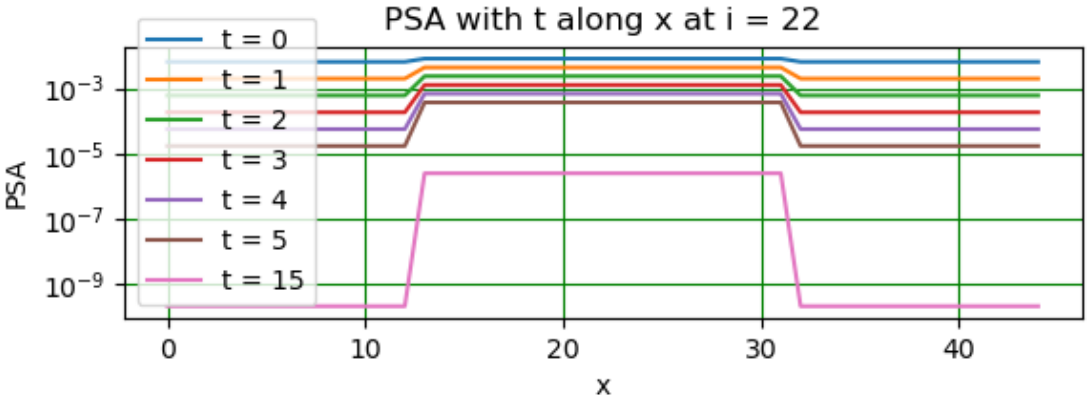


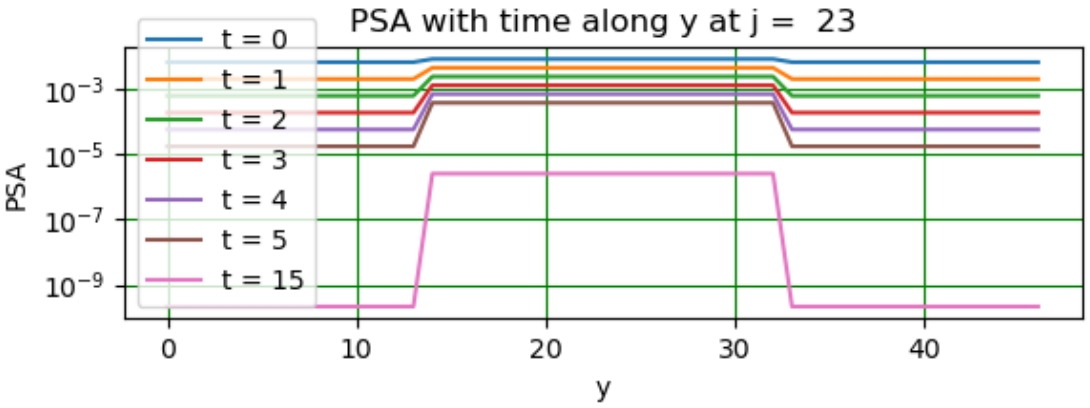

```
Diffusion PSA_top, no tumour
Diffusion converged, nt = 26, ext_test = 6.716970e-04, prostate_test =
2.245043e-04
```

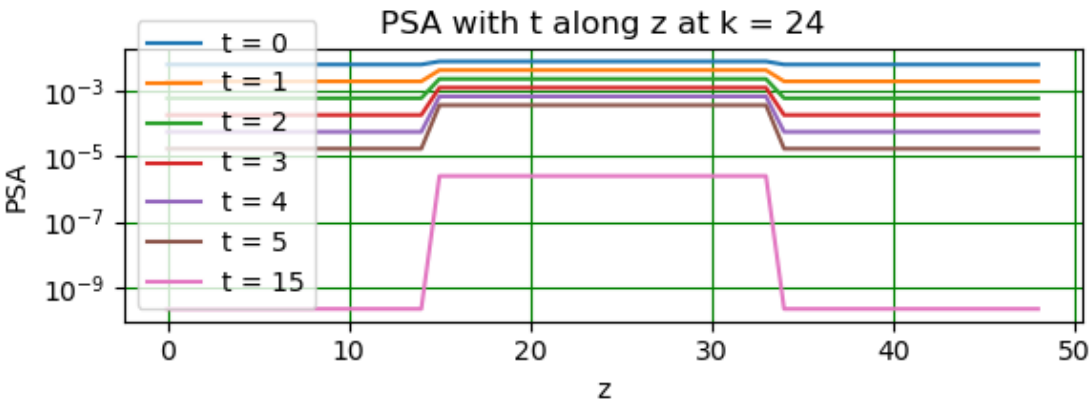

PSA, t = 26, xyz
1e−6
1.50
1.25
1.00
0.75
0.50
0.25
z
40
30
20
10
0
0
10
20
30
40
x
0
40
Blood
Prostate
Tumour


PSA, t = 26, plane at k = 24
y
x
Blood
Prostate
Tumour
PSA, t = 26, plane at j = 23
z
x
Blood
Prostate
Tumour
PSA, t = 26, plane at i = 22
z
y
Blood
Prostate
Tumour

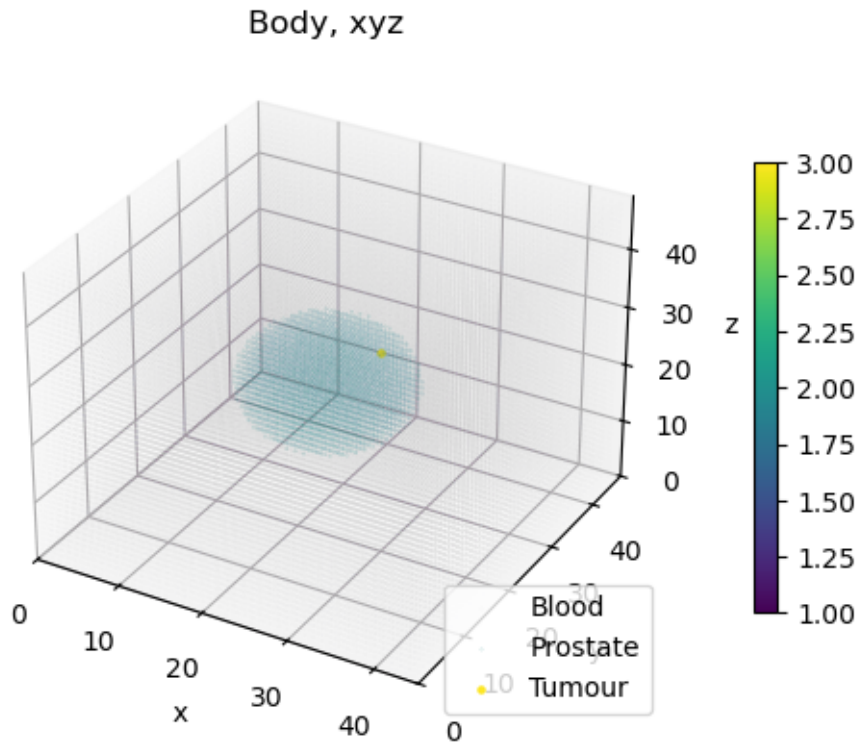

PSA with t at specified points
log(PSA)
t
(22, 23, 24)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

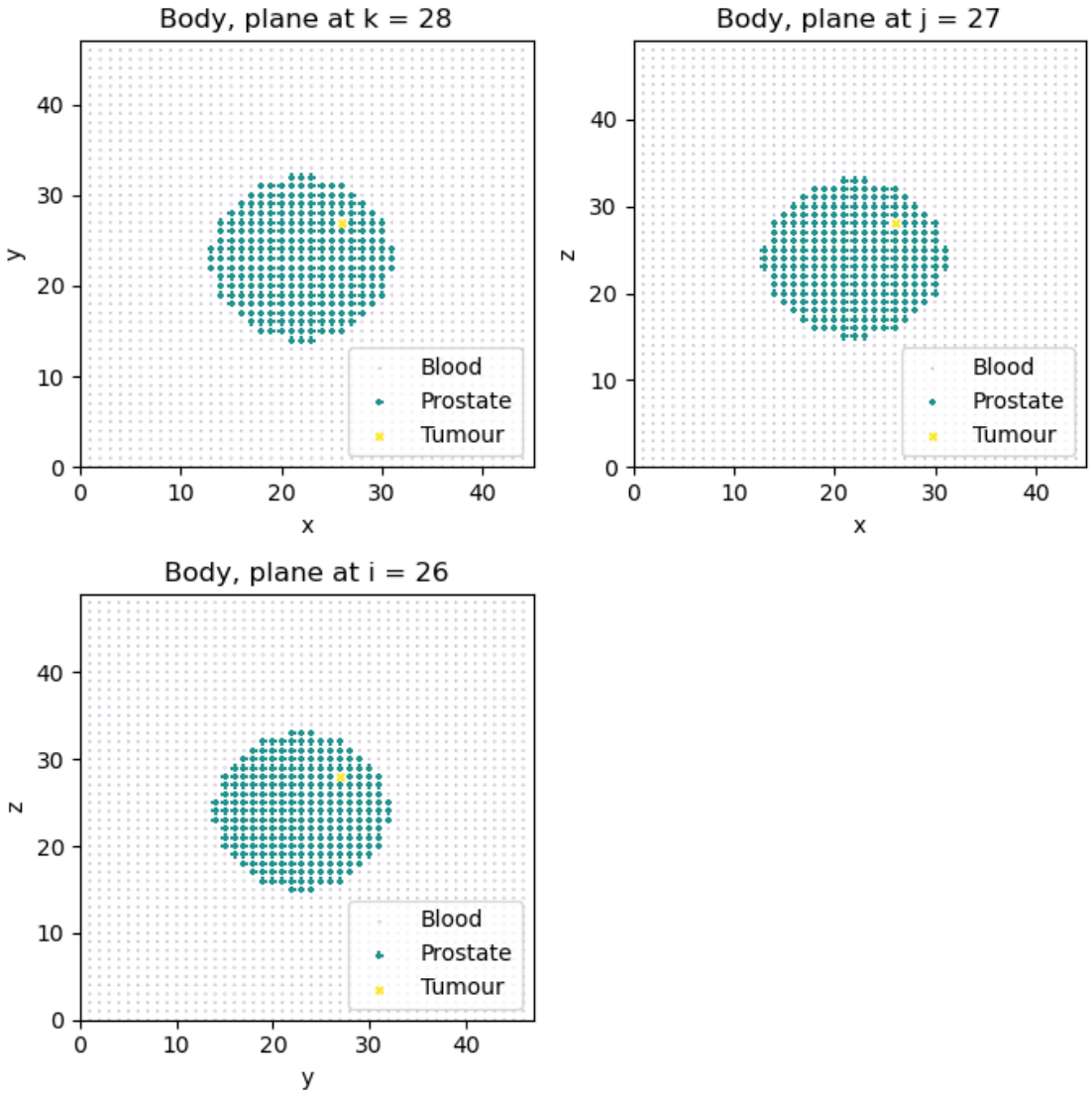

PSA with t along x at i = 22
PSA
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5
t = 15


PSA with time along y at j = 23
PSA
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5
t = 15

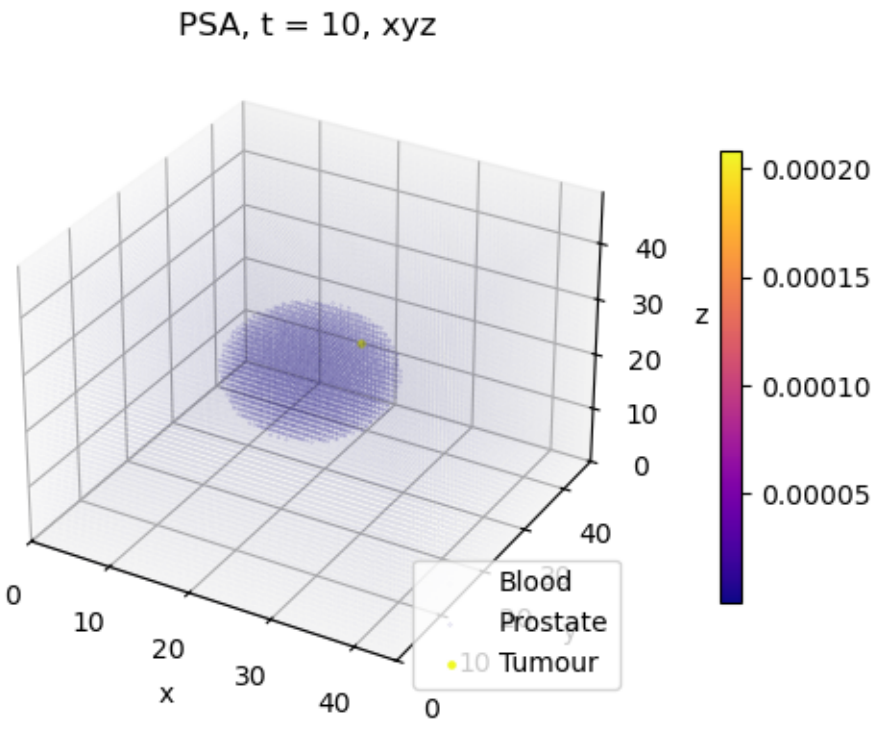

PSA with t along z at k = 24
PSA
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5
t = 15

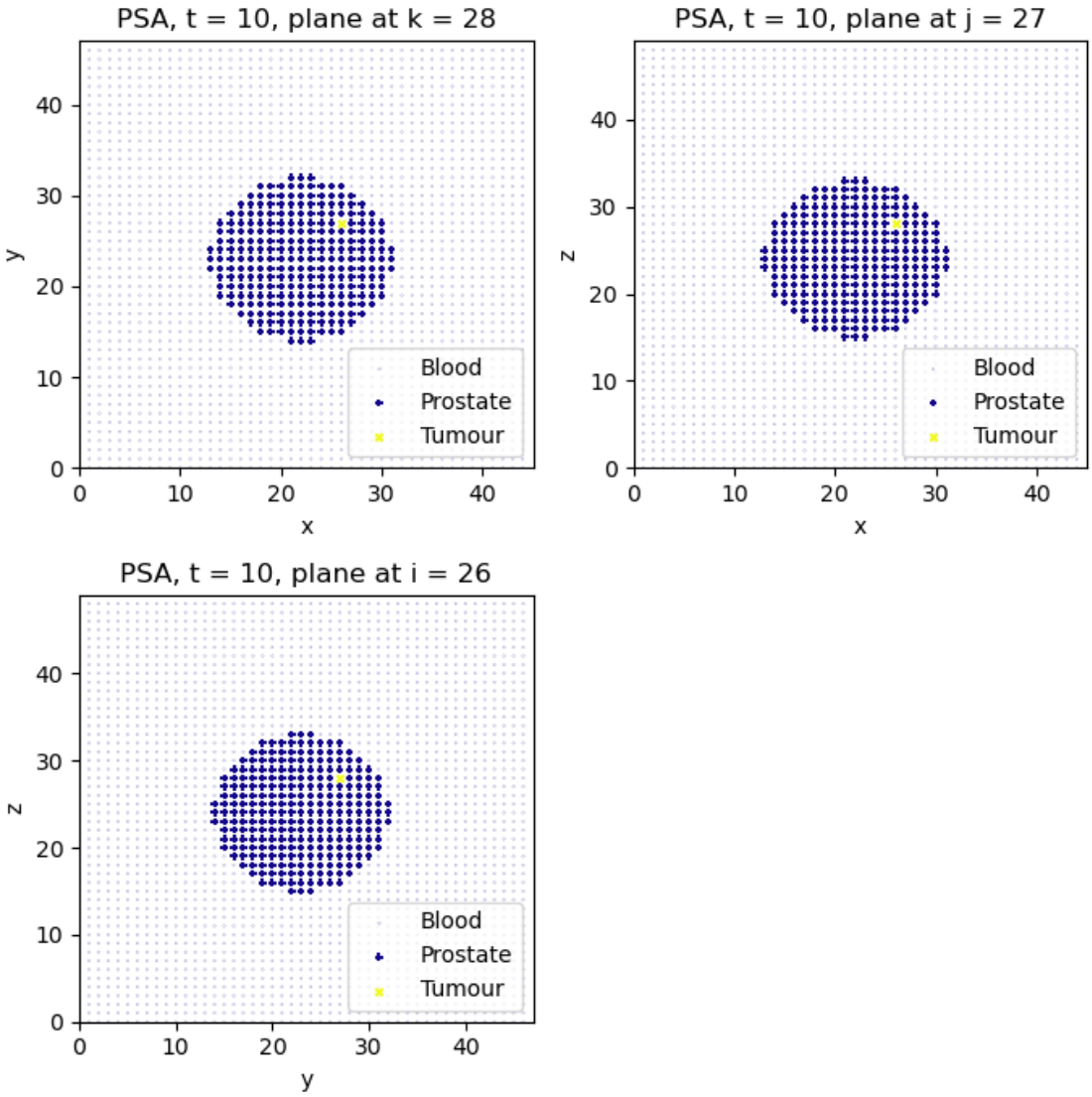
```
At time 0:
No. tumour cells 0, prostate cells 4139, blood cells 99496
PSA_top_level 4.105015e-11
--------------------------------------------------------------------------------
----------------------------
Time 1

Number of blood cells is 99496
Number of cells in prostate is 4138
Number of cells in tumour is 1
```

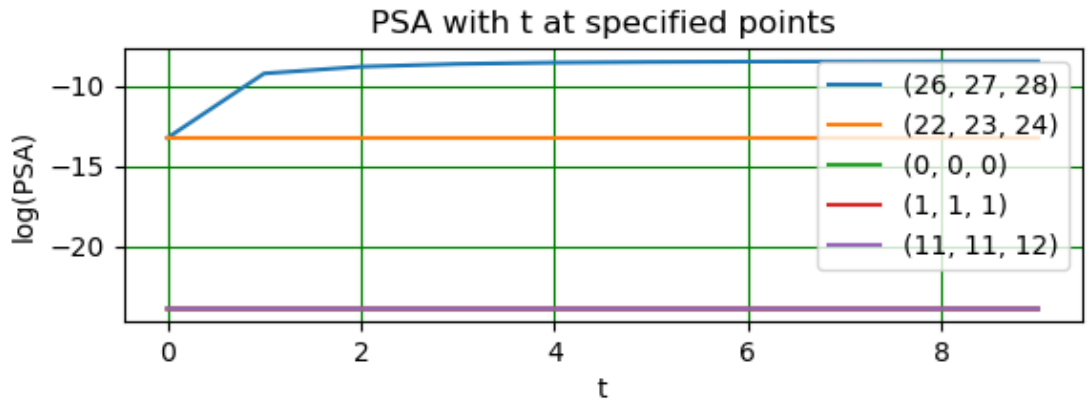


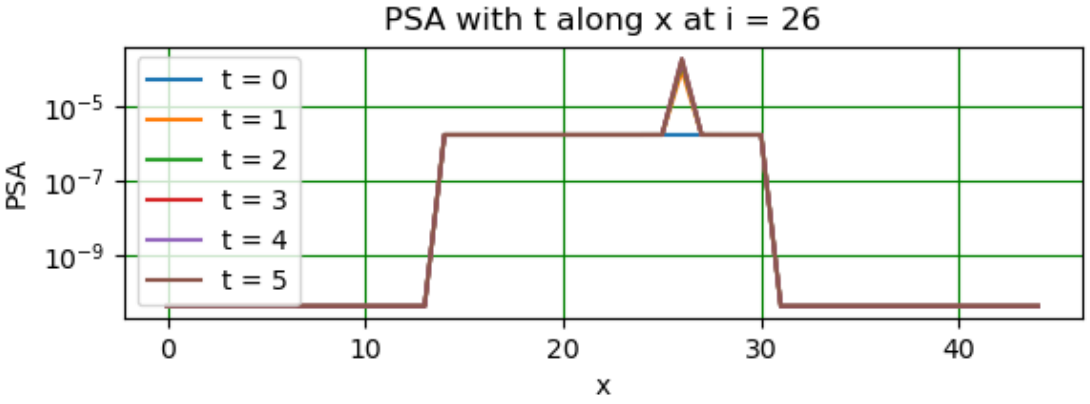


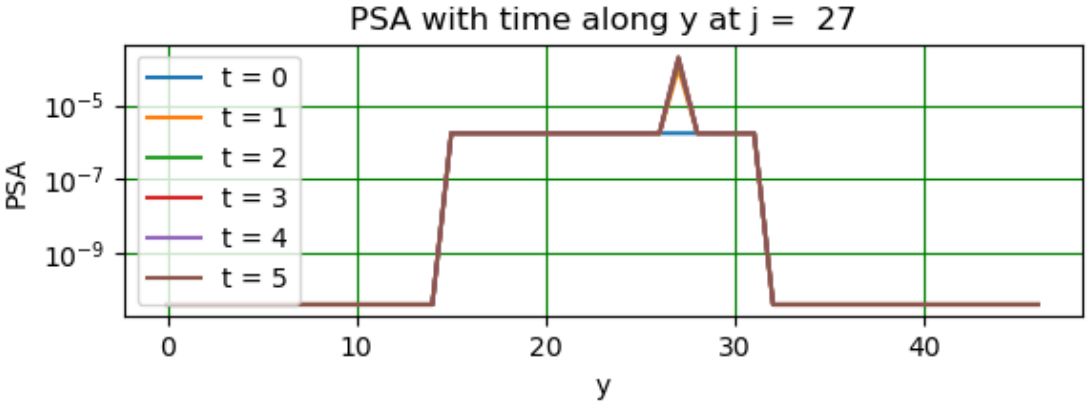

```
Diffusion PSA_top, with initial tumour
Diffusion converged, nt = 10, ext_test = 7.471879e-07, prostate_test =
9.488810e-04
```

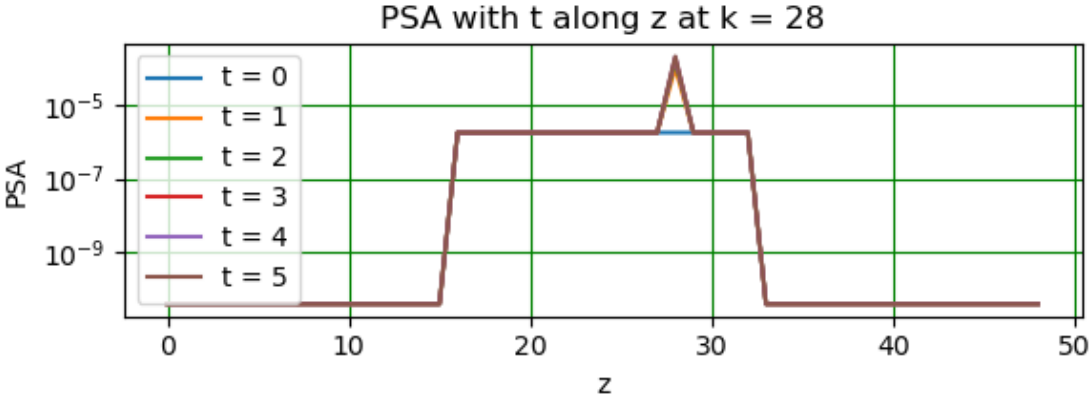

PSA, t = 10, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

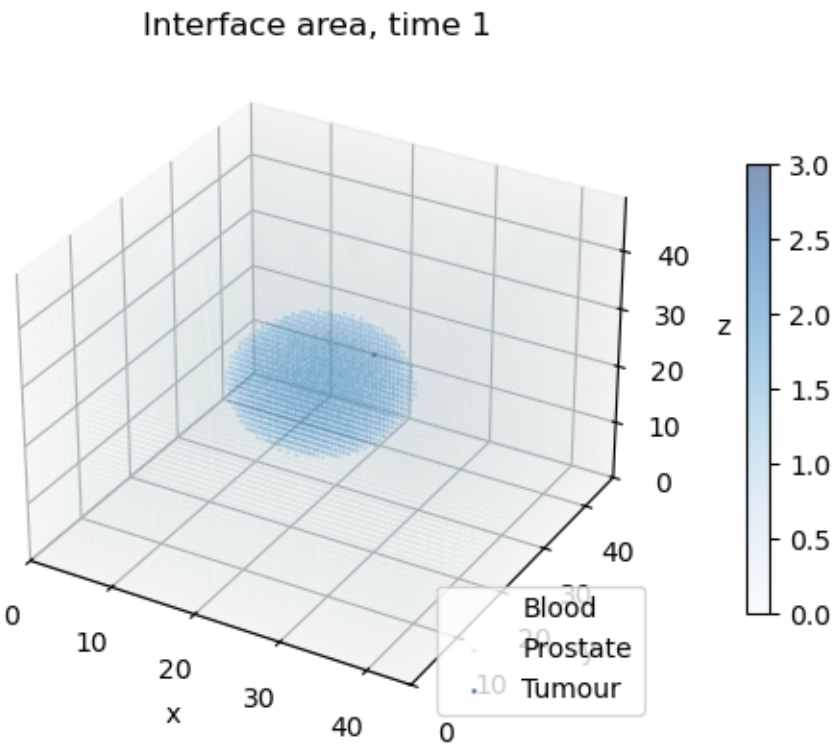

PSA, t = 10, plane at k = 28
PSA, t = 10, plane at j = 27
PSA, t = 10, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0
2
4
6
8
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

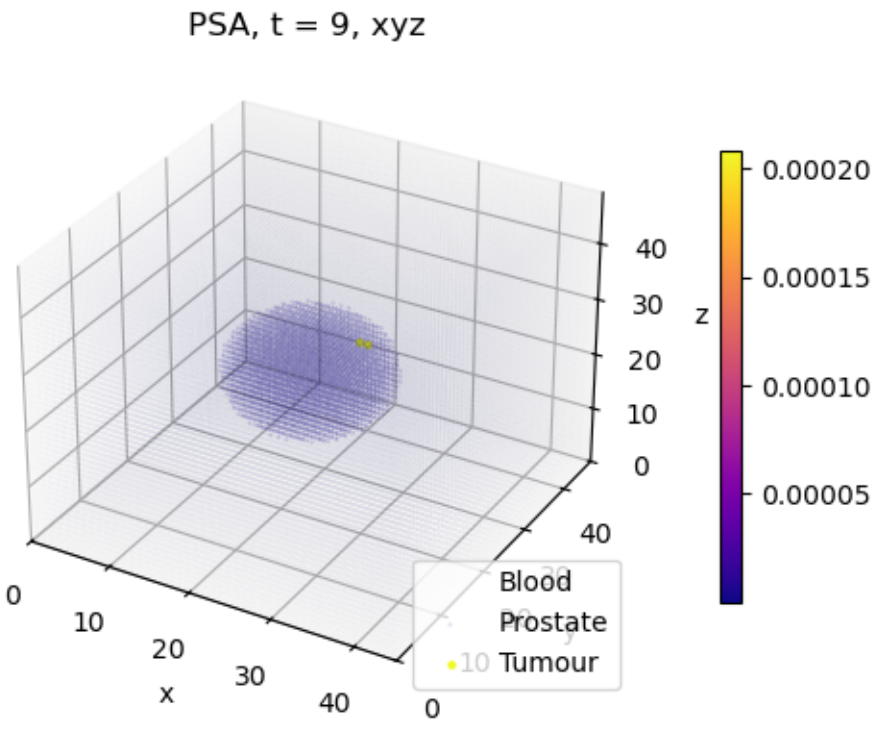

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

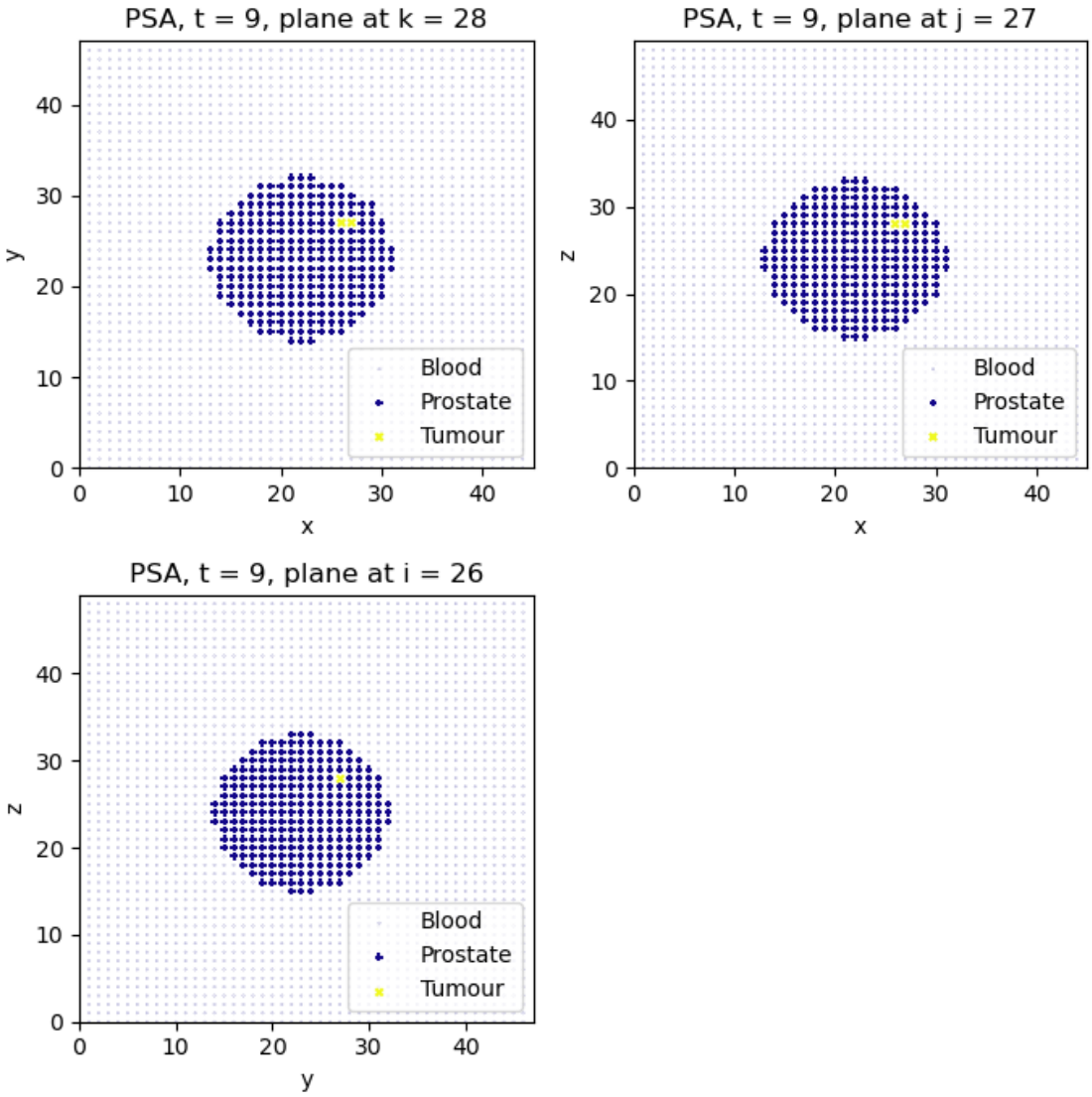

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

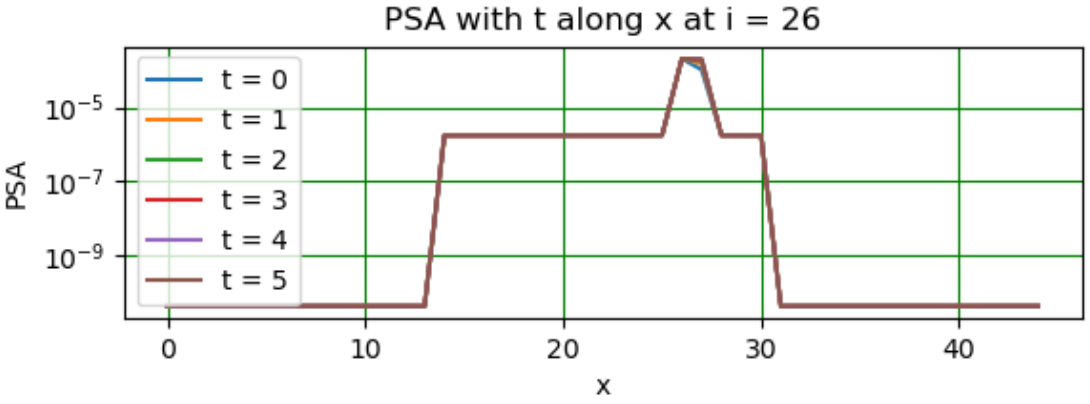


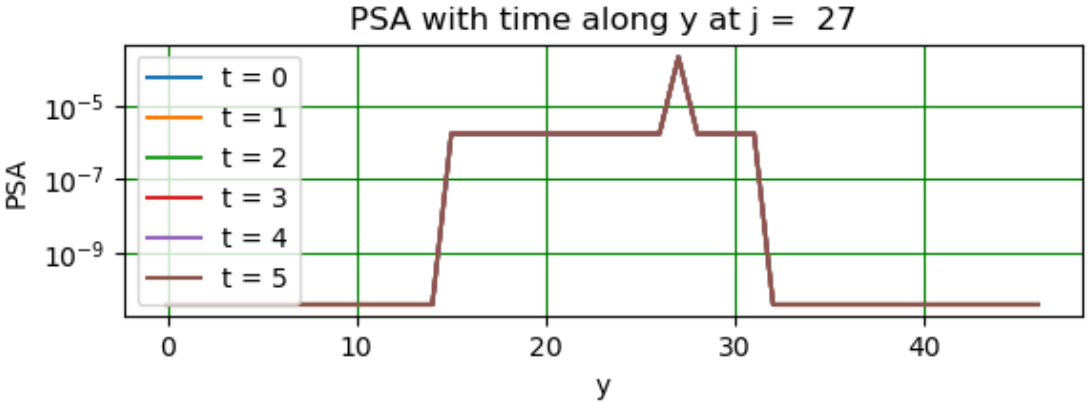

```
At time 1:
No. tumour cells 0, prostate cells 4139, blood cells 99496
PSA_top_level 4.105015e-11
-------------------------------------------------------------------------------
--------------------------
Time 2
Diffusion PSA_top, with tumour
Diffusion converged, nt = 9, ext_test = 1.627483e-08, prostate_test =
8.809597e-04
```

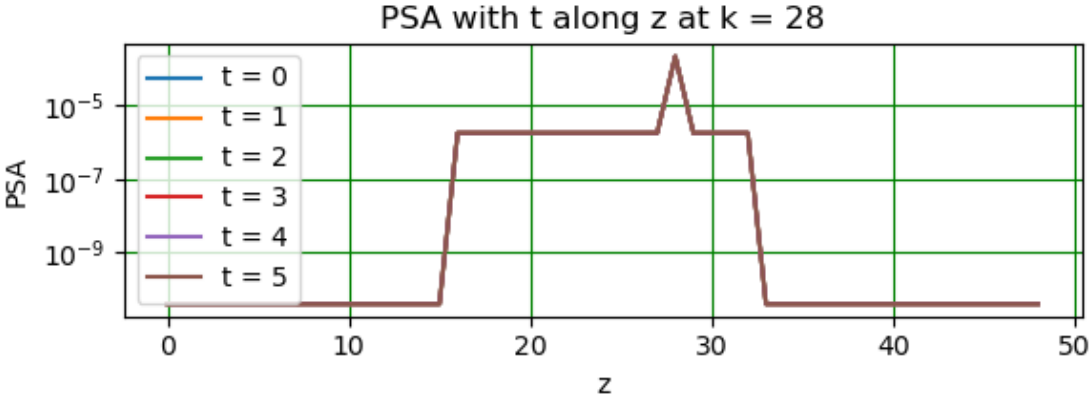

PSA, t = 9, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

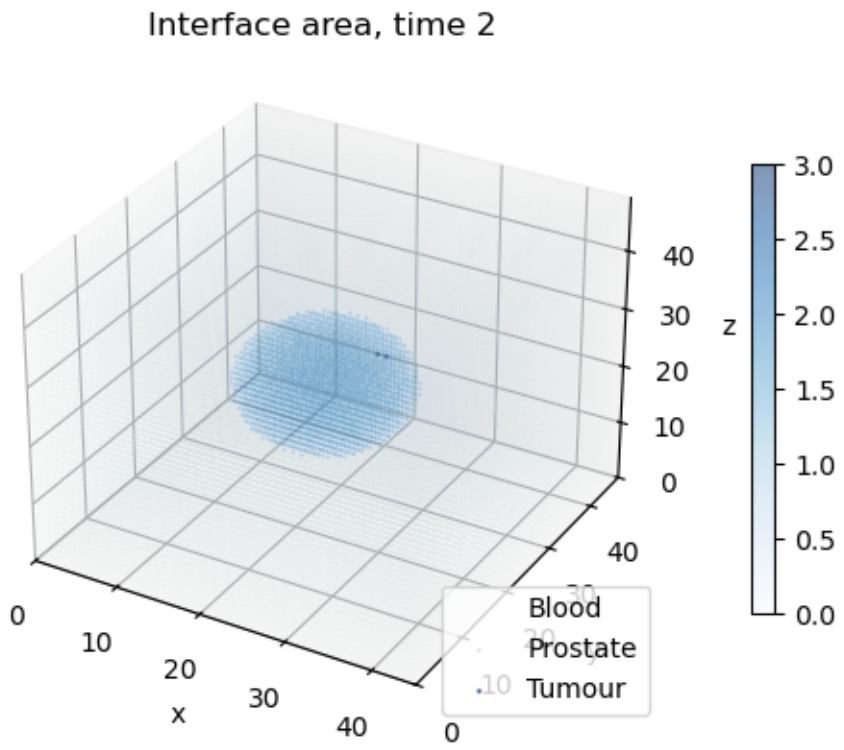

PSA, t = 9, plane at k = 28
PSA, t = 9, plane at j = 27
PSA, t = 9, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
7
8
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

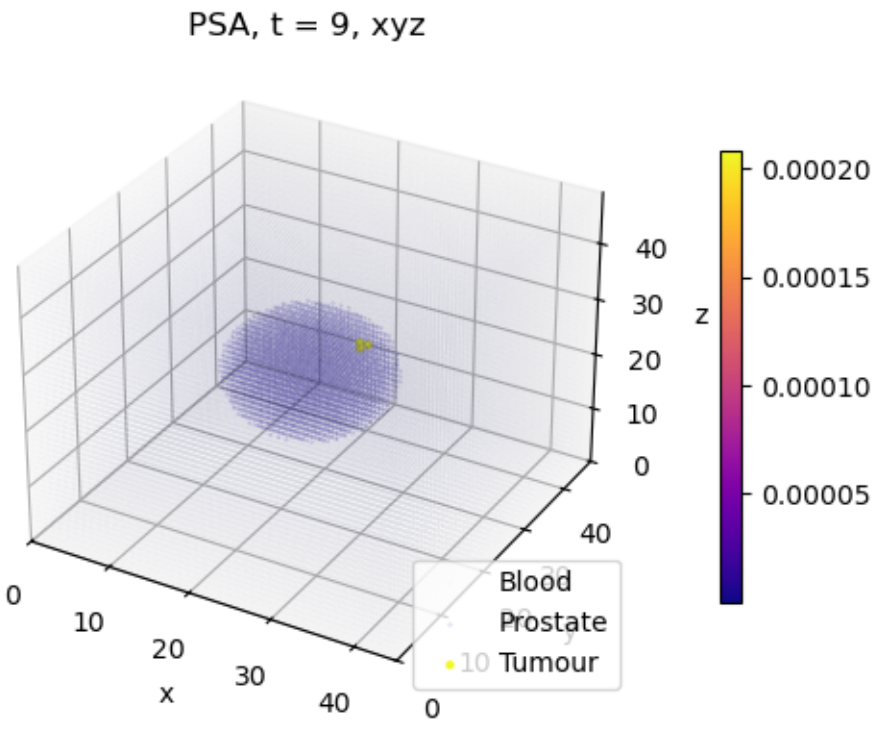

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

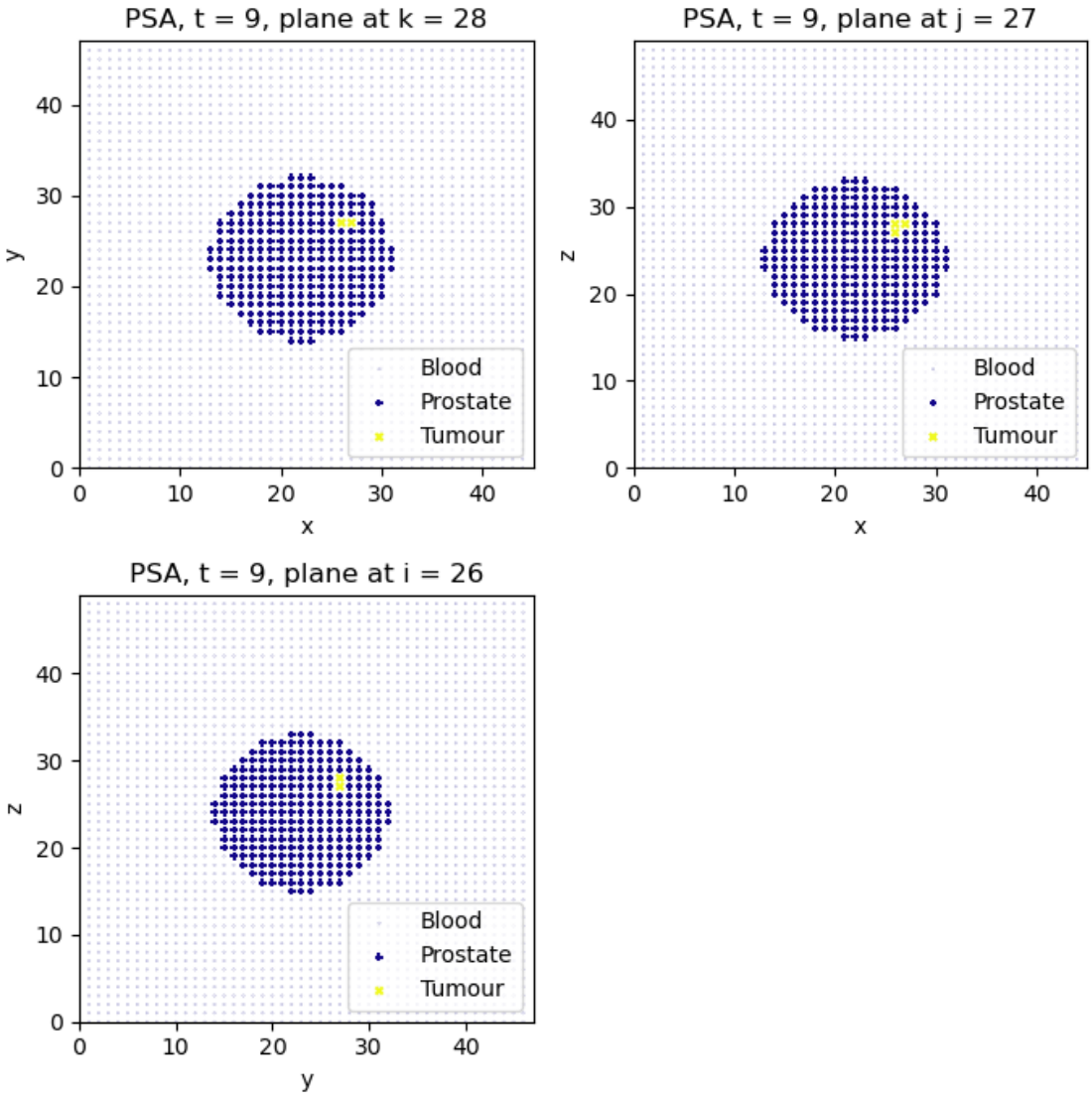

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

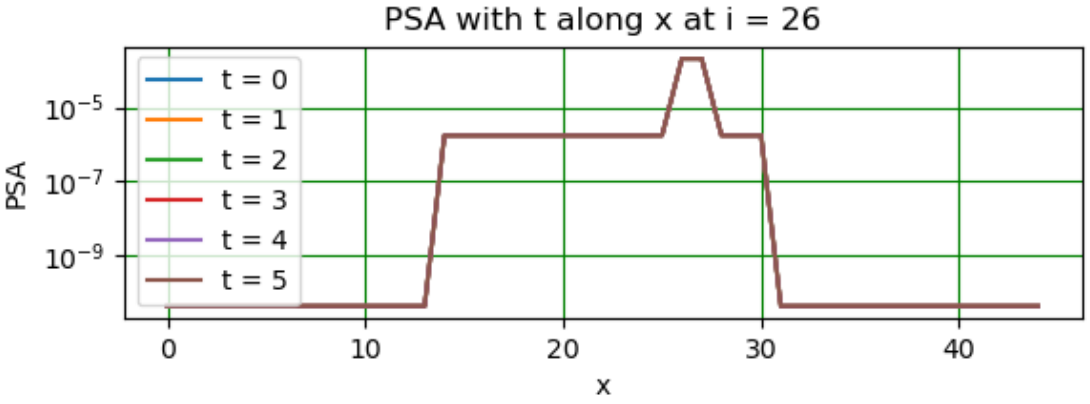


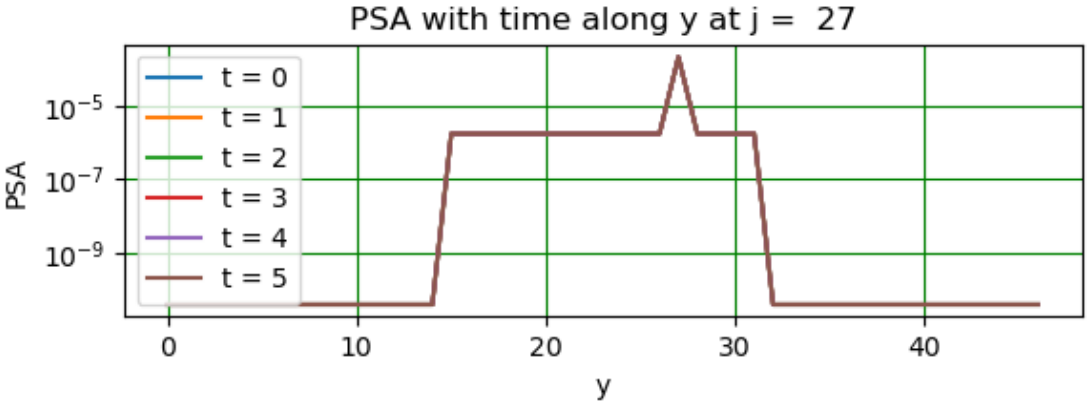

```
At time 2:
No. tumour cells 2, prostate cells 4138, blood cells 99495
PSA_top_level 4.098137e-11
-------------------------------------------------------------------------------
--------------------------
Time 3
Diffusion PSA_top, with tumour
Diffusion converged, nt = 9, ext_test = 3.765909e-10, prostate_test =
5.870463e-04
```


PSA, t = 9, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

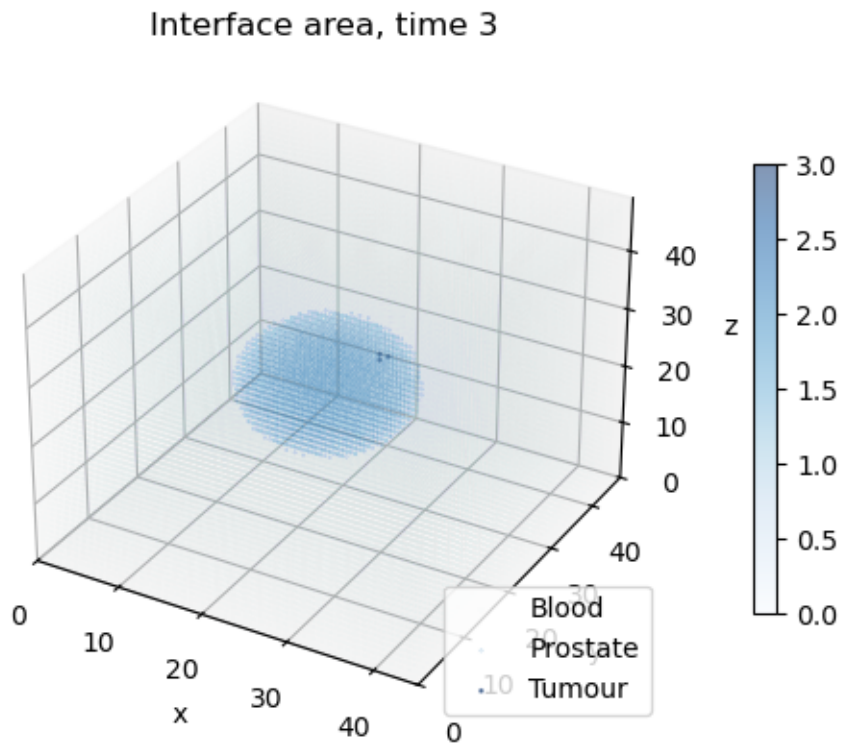

PSA, t = 9, plane at k = 28
PSA, t = 9, plane at j = 27
PSA, t = 9, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
7
8
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

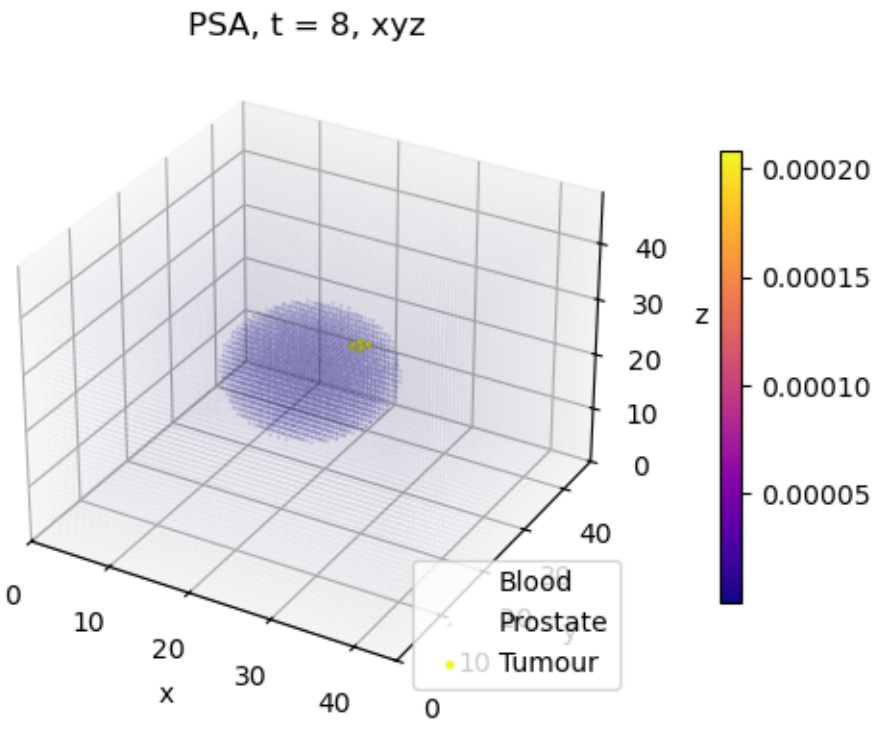

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

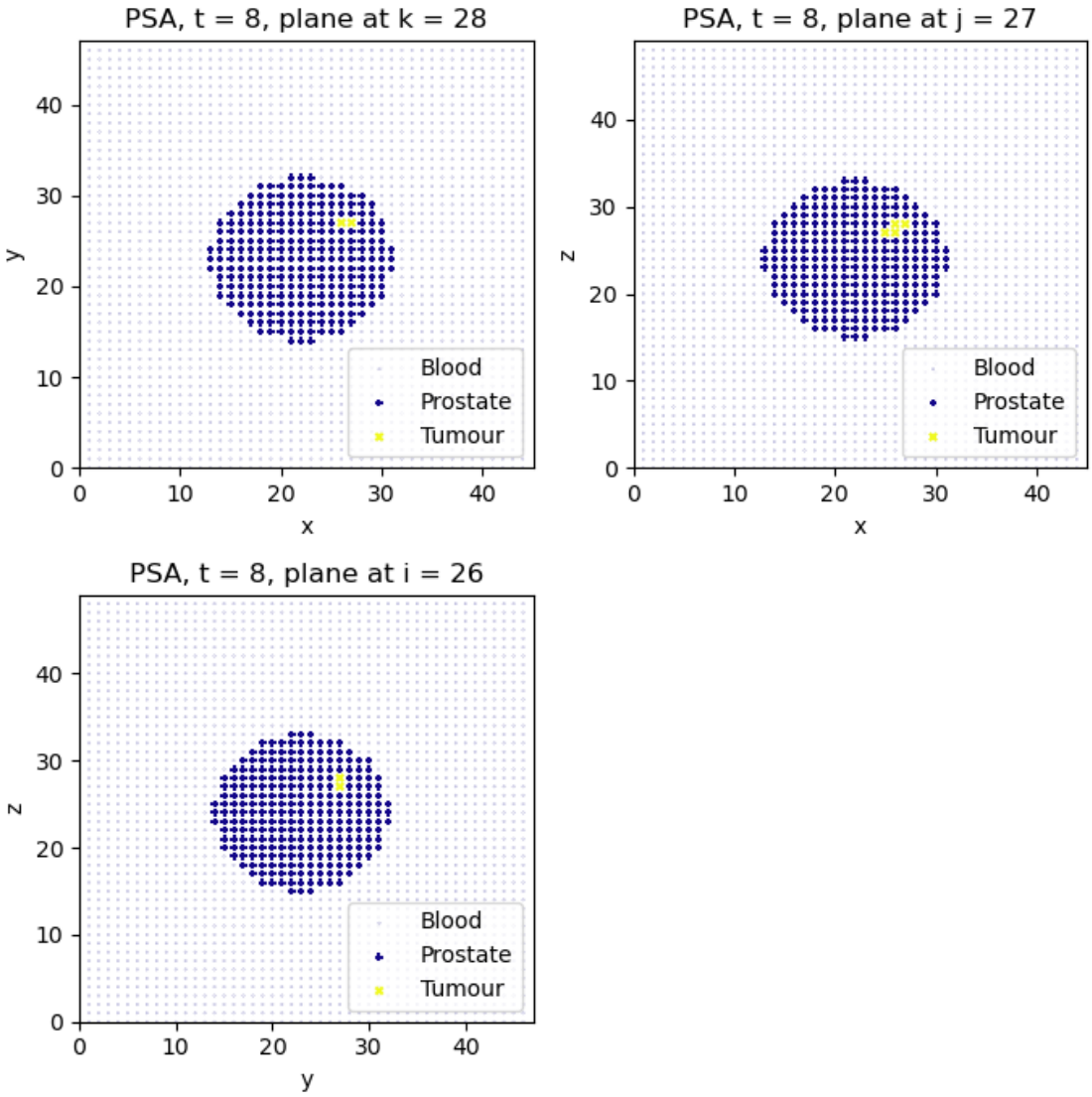

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

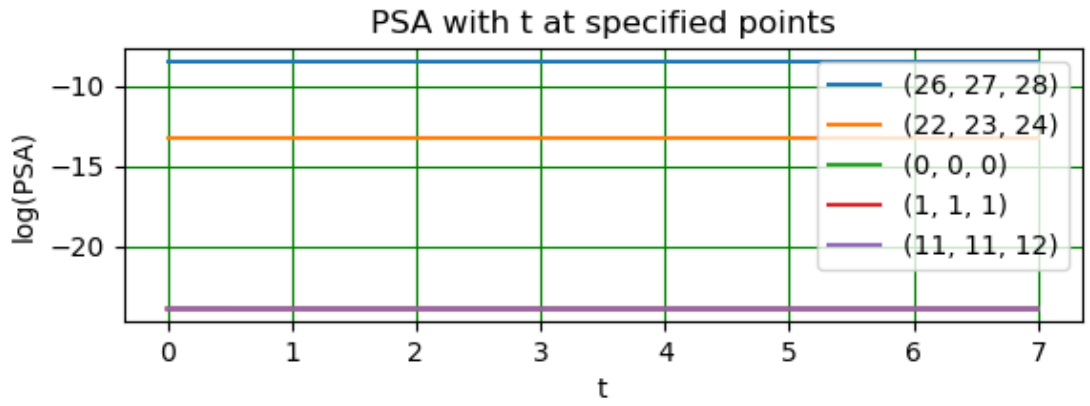

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

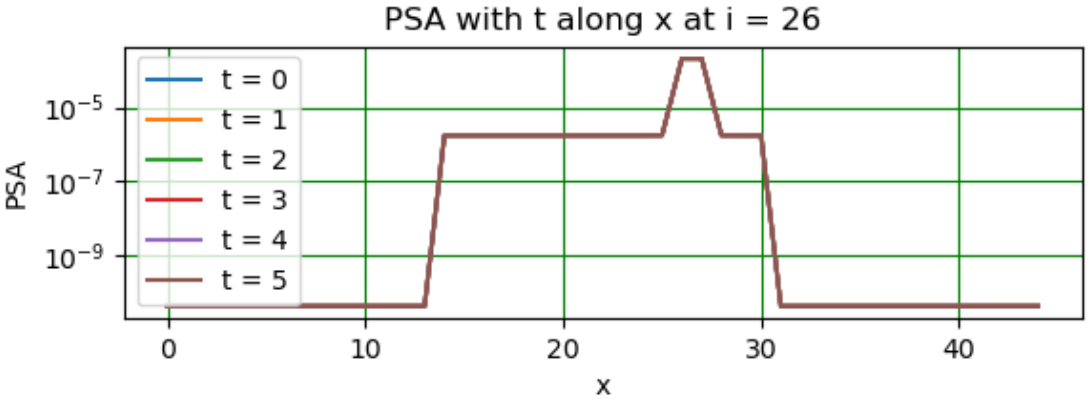


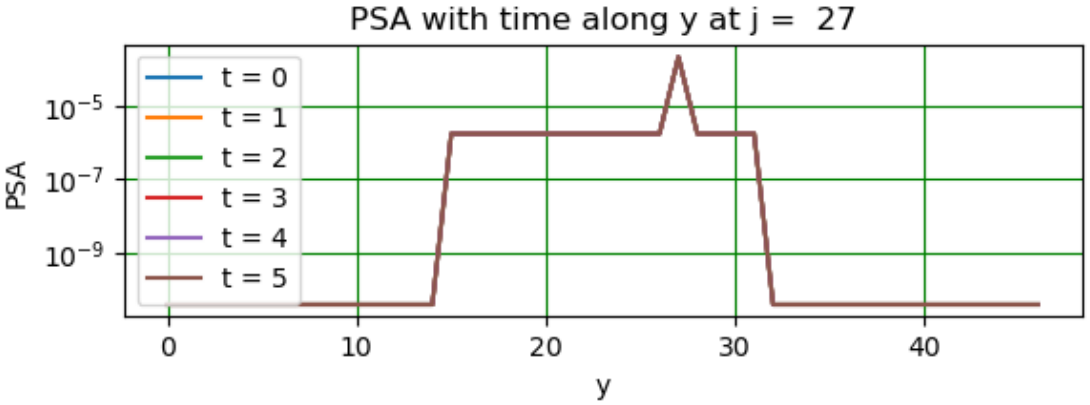

```
At time 3:
No. tumour cells 3, prostate cells 4138, blood cells 99494
PSA_top_level 4.098137e-11
--------------------------------------------------------------------------------
--------------------------
Time 4
Diffusion PSA_top, with tumour
Diffusion converged, nt = 8, ext_test = 4.770726e-08, prostate_test =
8.166066e-04
```


PSA, t = 8, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

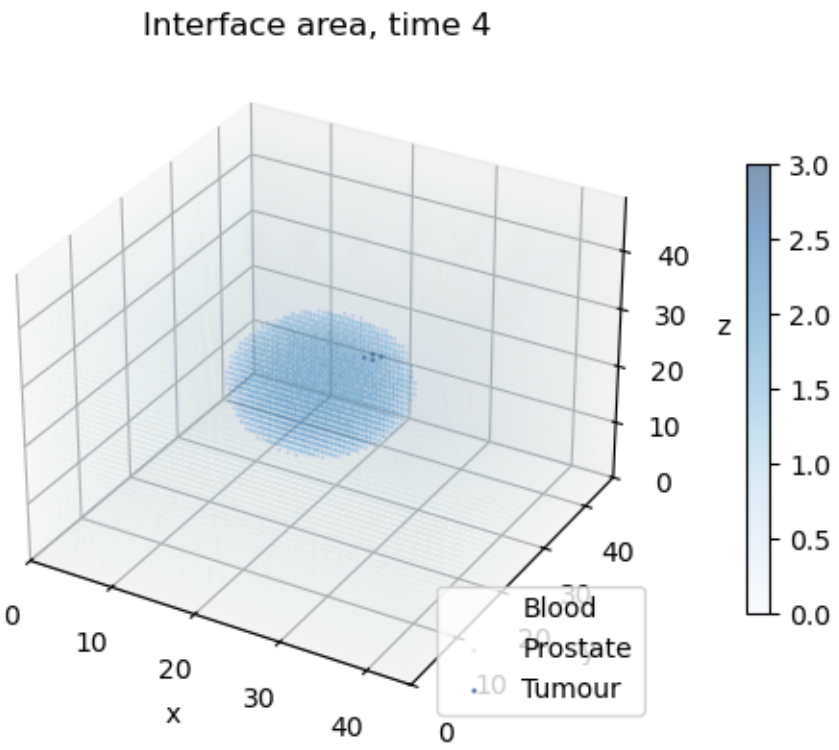

PSA, t = 8, plane at k = 28
PSA, t = 8, plane at j = 27
PSA, t = 8, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
7
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

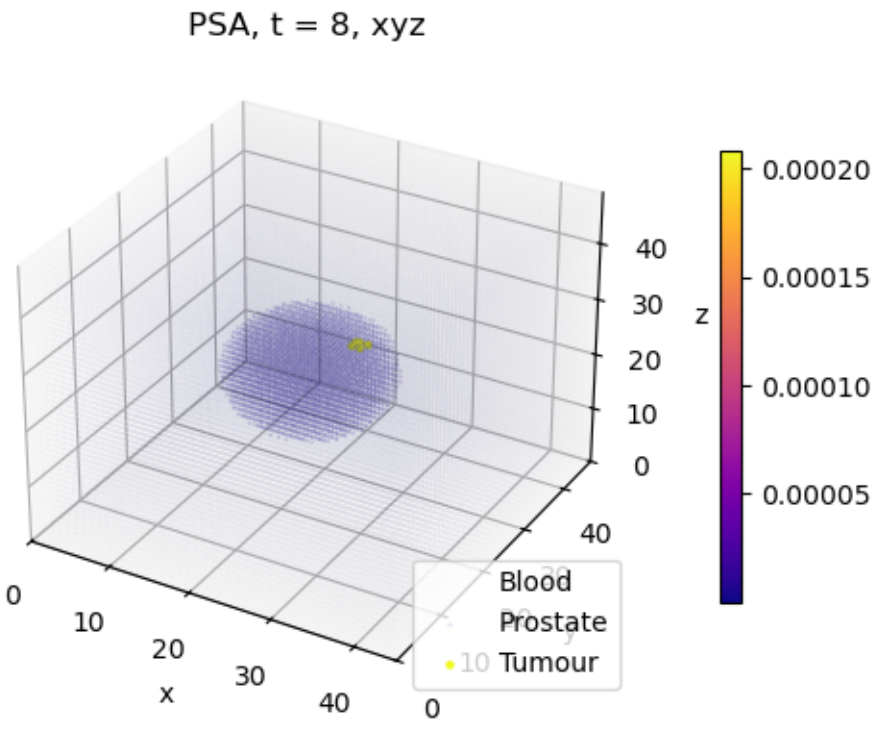

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

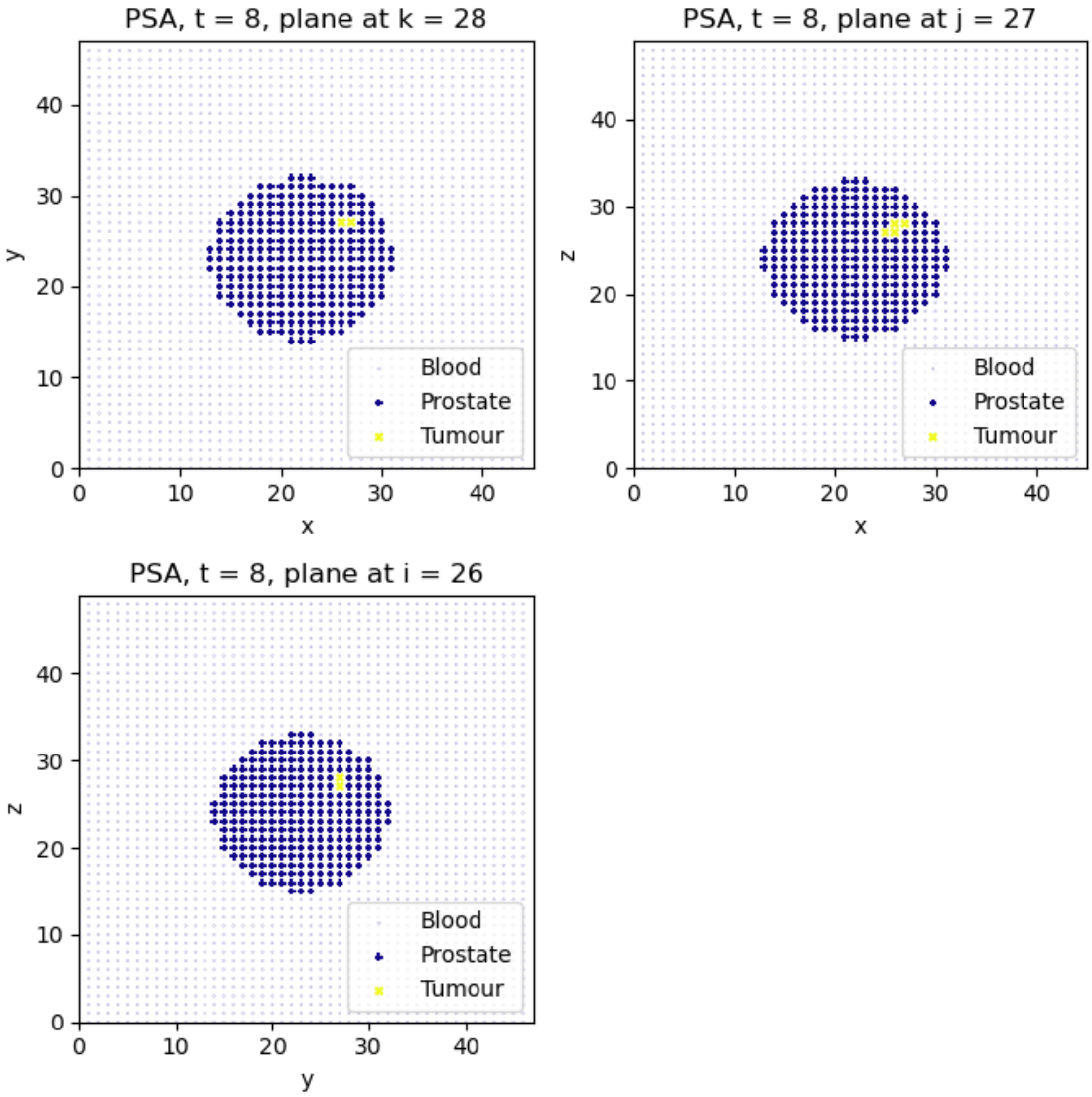

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

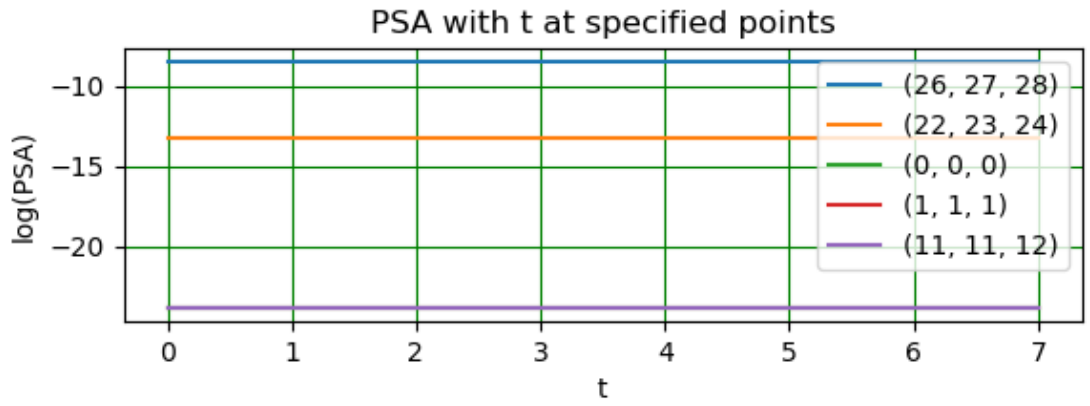

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

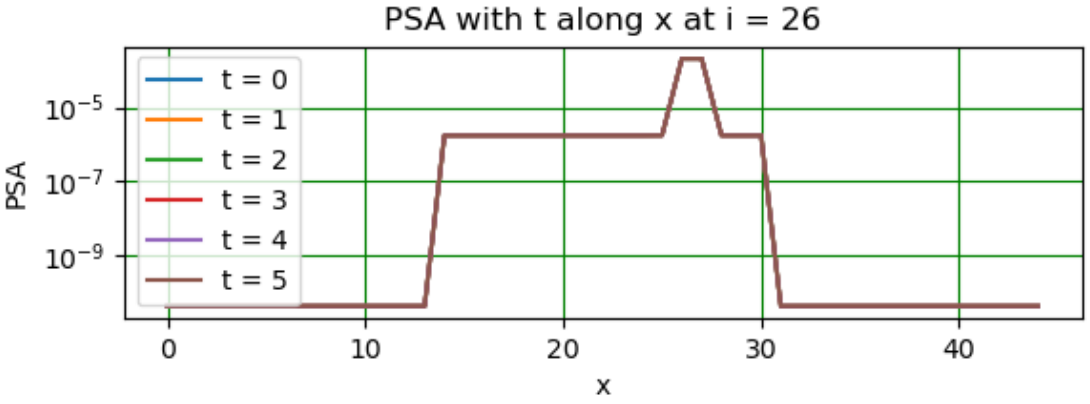


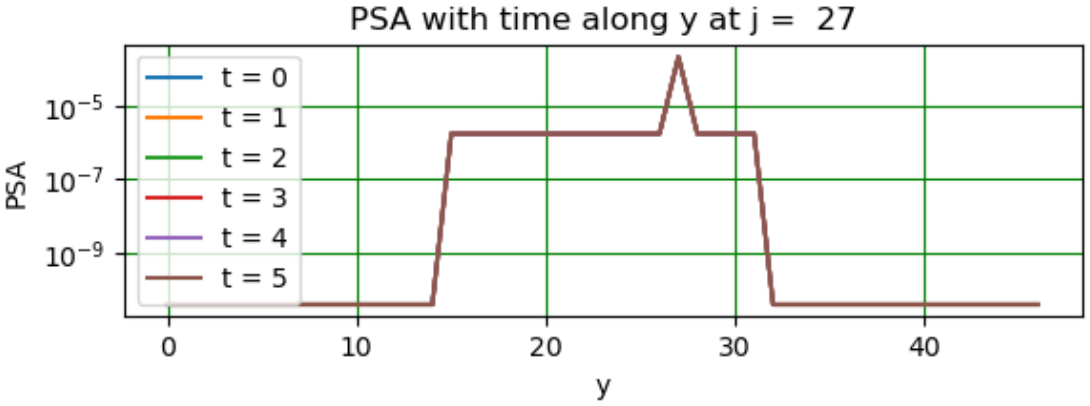

```
At time 4:
No. tumour cells 4, prostate cells 4138, blood cells 99493
PSA_top_level 4.094478e-11
-------------------------------------------------------------------------------
--------------------------
Time 5
Diffusion PSA_top, with tumour
Diffusion converged, nt = 8, ext_test = 5.732670e-10, prostate_test =
6.533235e-04
```


PSA, t = 8, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

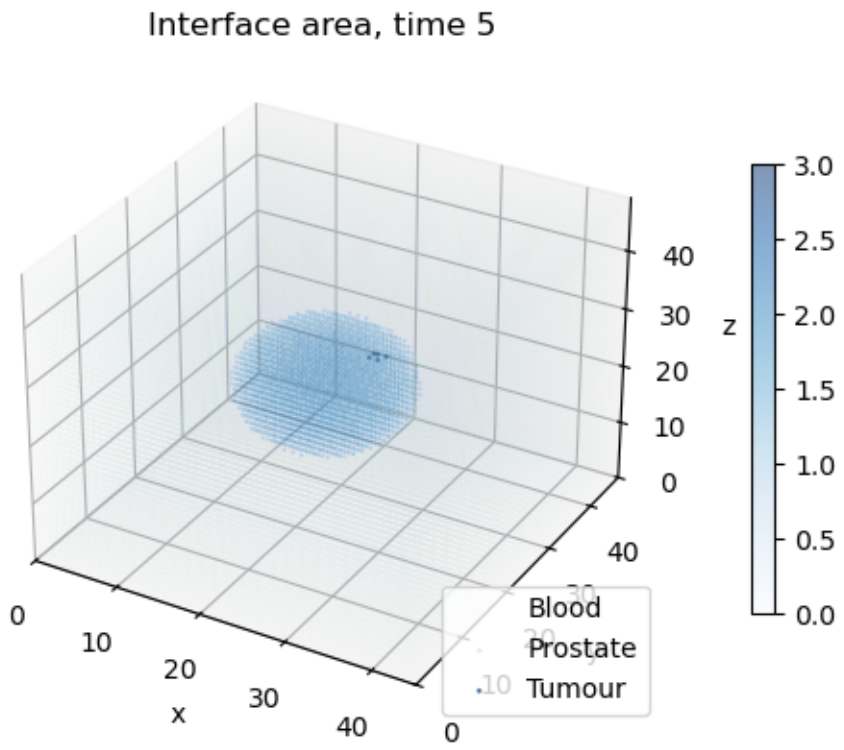

PSA, t = 8, plane at k = 28
PSA, t = 8, plane at j = 27
PSA, t = 8, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

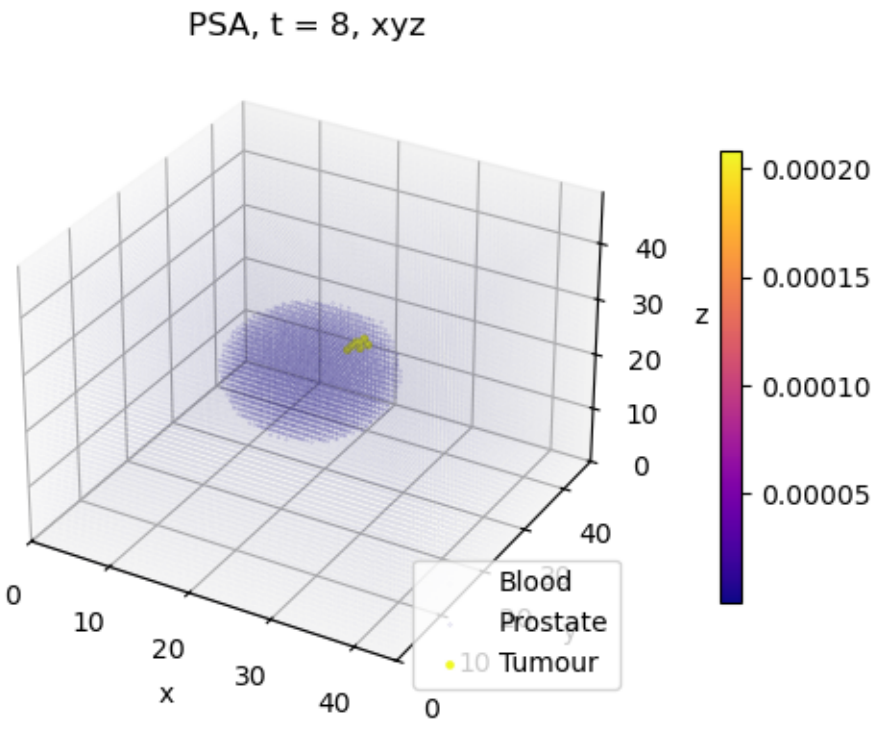

PSA with t along x at i = 26
PSA
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

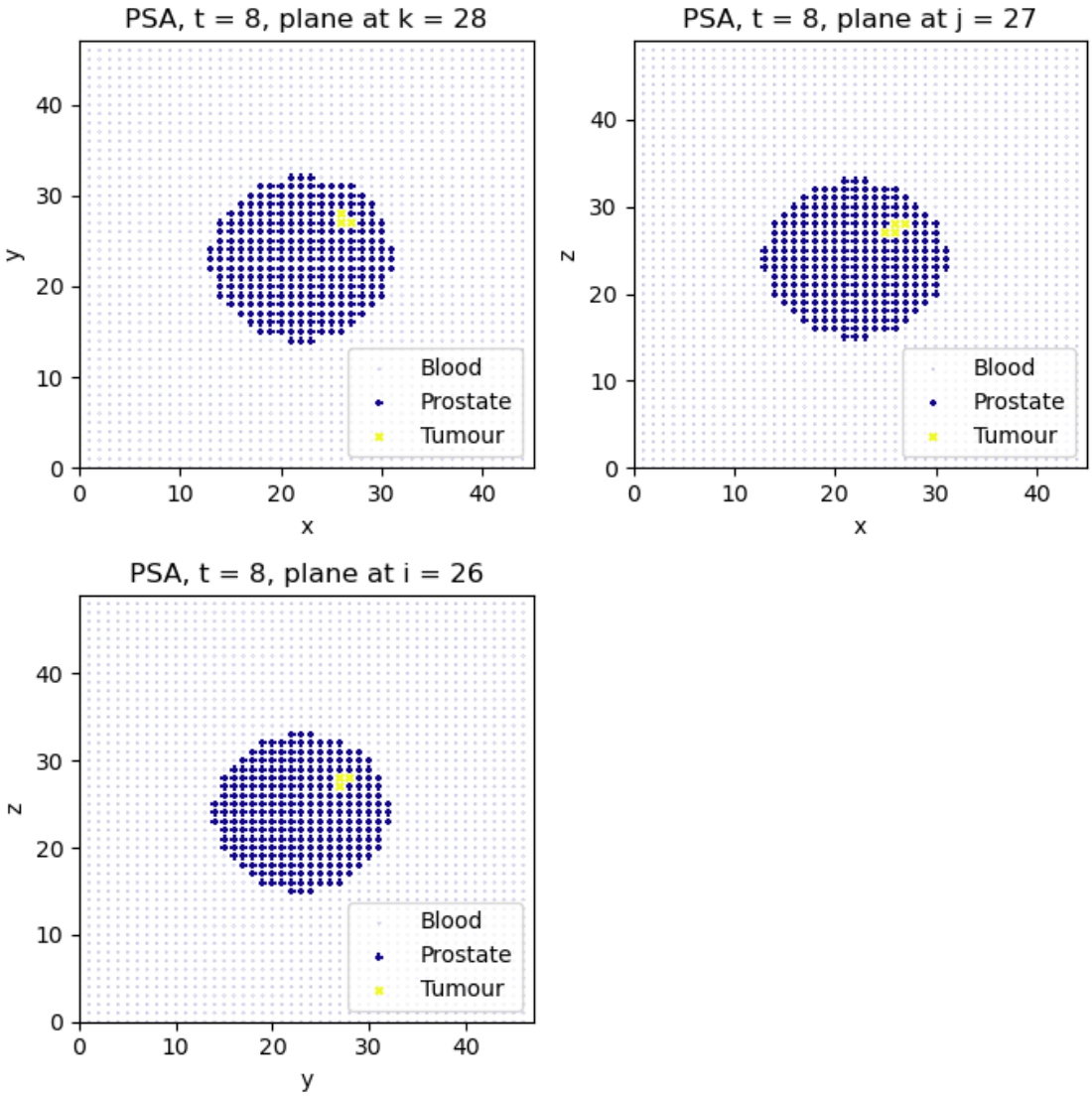

PSA with time along y at j = 27
PSA
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

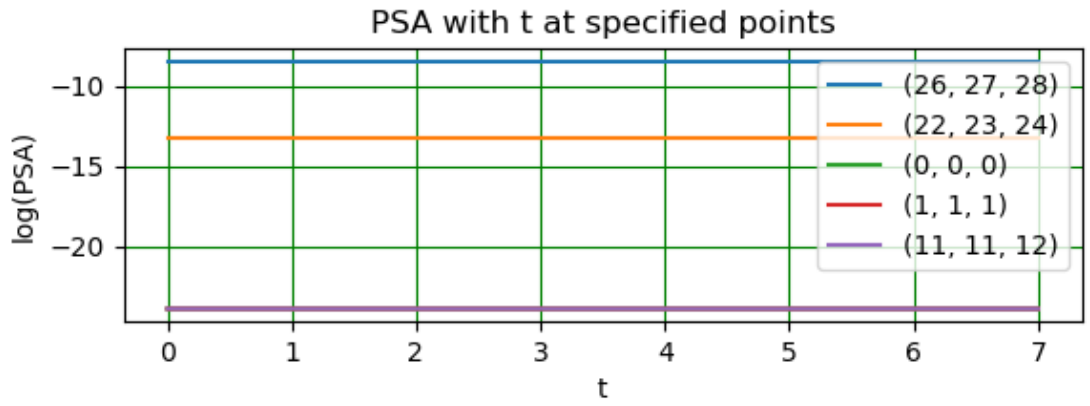

PSA with t along z at k = 28
PSA
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

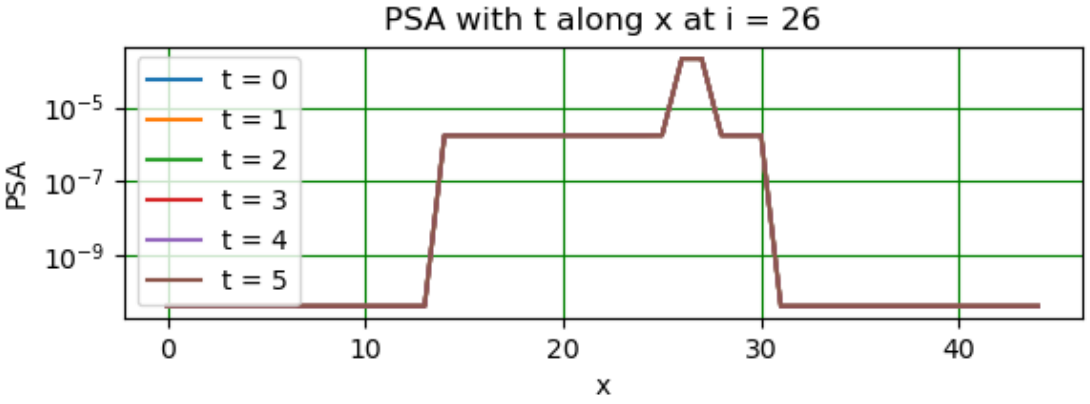


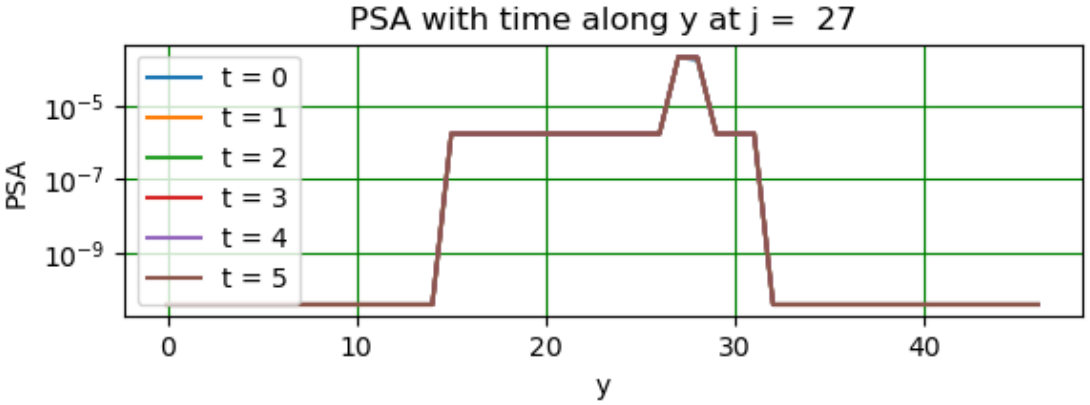

```
At time 5:
No. tumour cells 5, prostate cells 4138, blood cells 99492
PSA_top_level 4.094478e-11
-------------------------------------------------------------------------------
--------------------------
Time 6
Diffusion PSA_top, with tumour
Diffusion converged, nt = 8, ext_test = 9.521548e-08, prostate_test =
5.442506e-04
```


PSA, t = 8, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

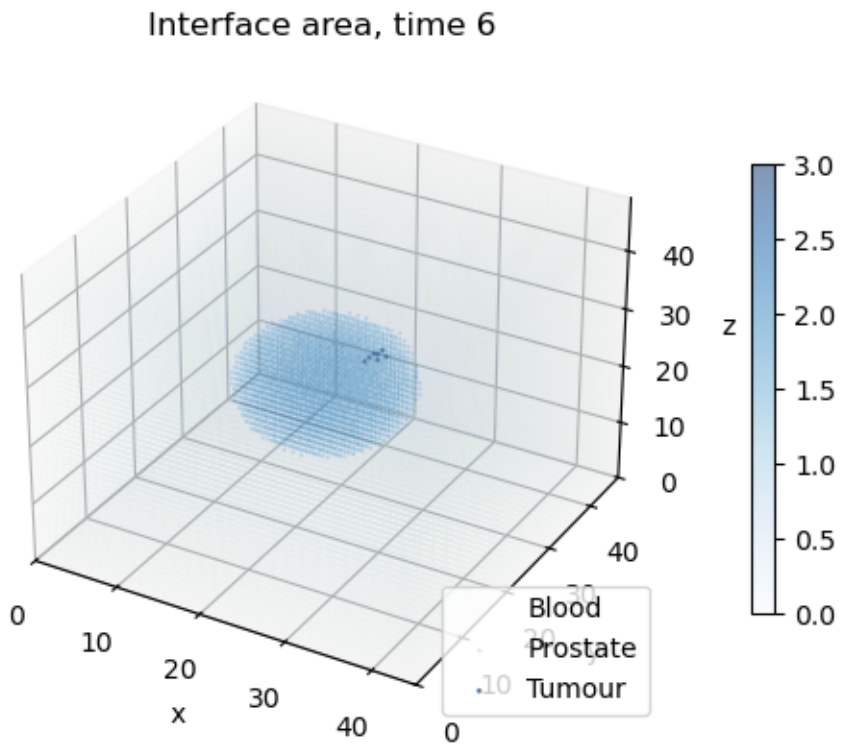

PSA, t = 8, plane at k = 28
PSA, t = 8, plane at j = 27
PSA, t = 8, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
7
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

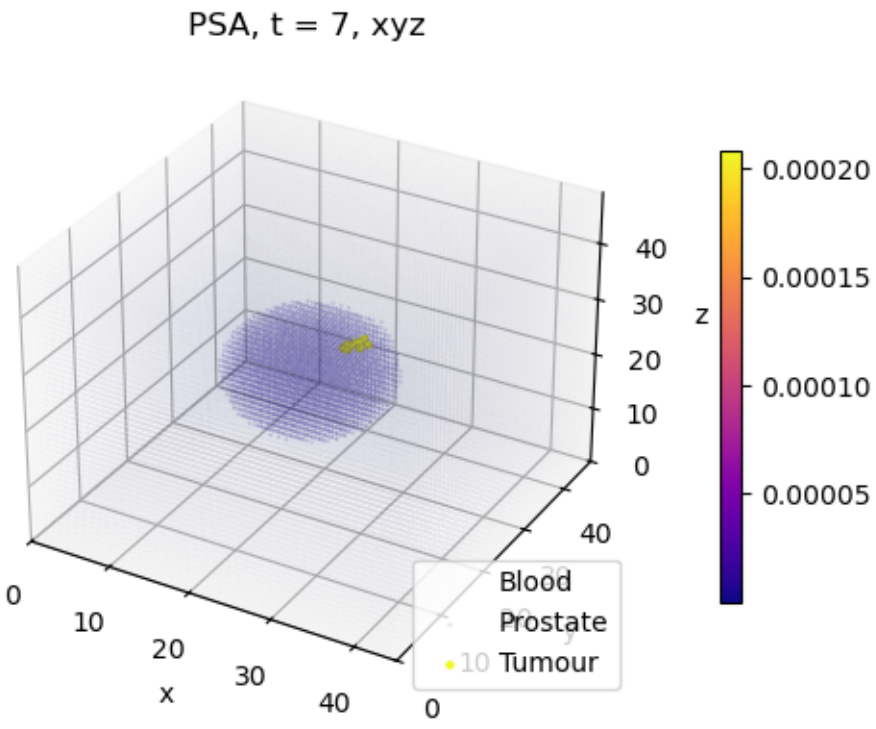

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

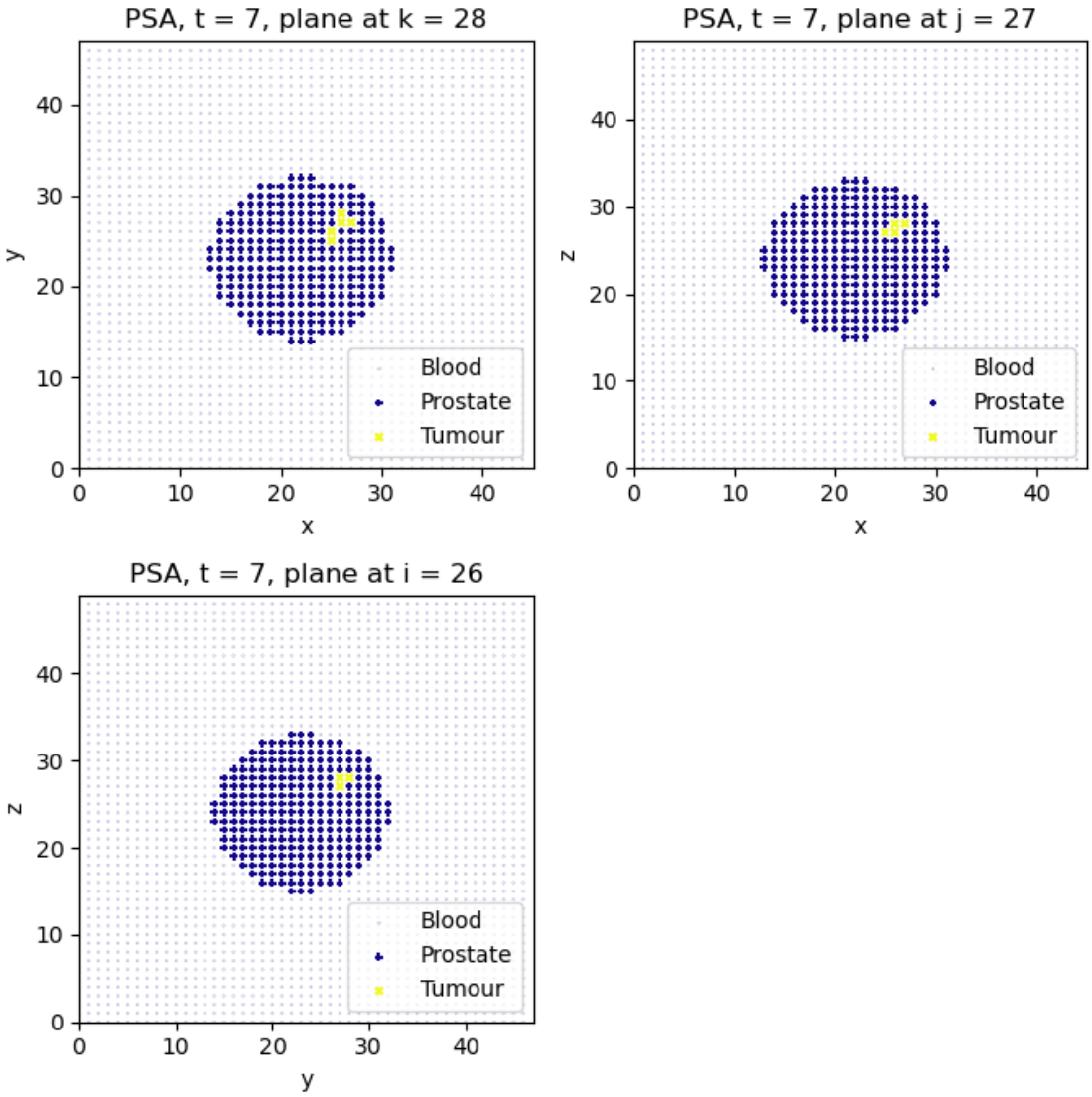

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

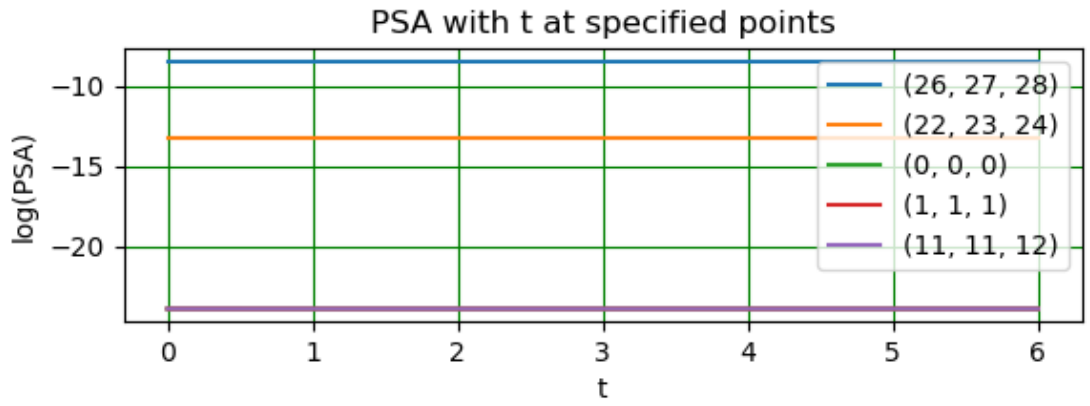

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

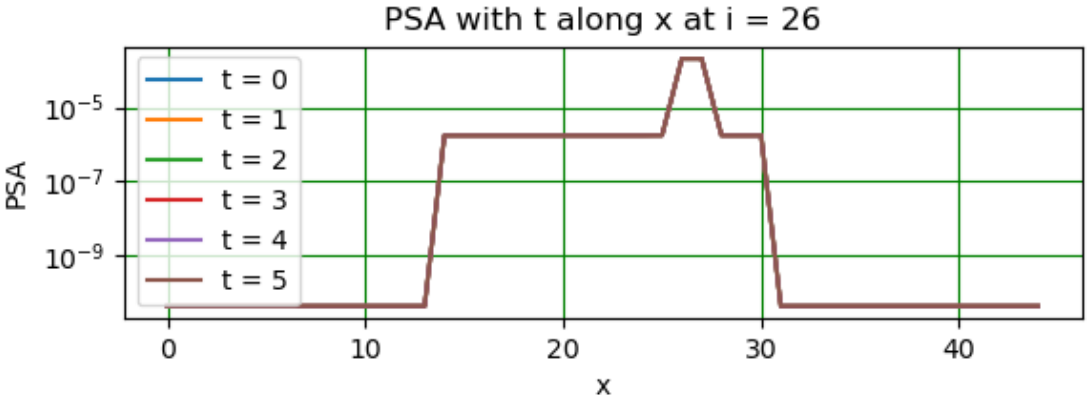


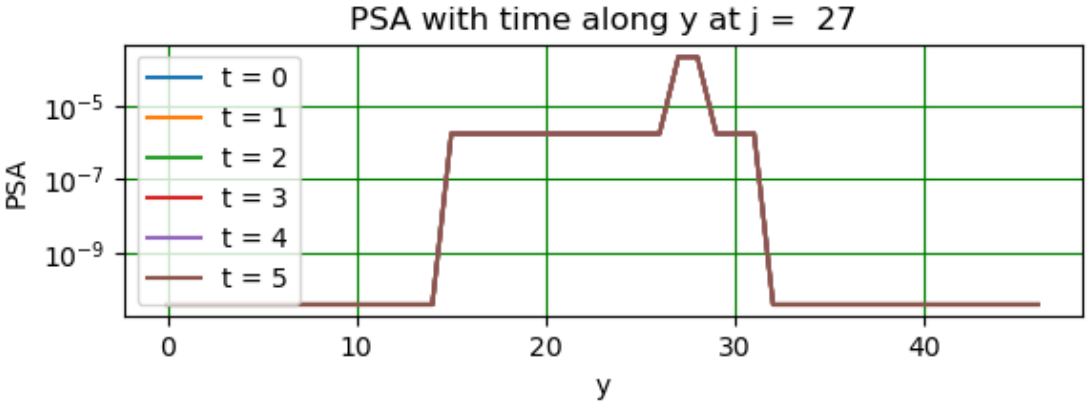

```
At time 6:
No. tumour cells 7, prostate cells 4138, blood cells 99490
PSA_top_level 4.087180e-11
--------------------------------------------------------------------------------
--------------------------
Time 7
Diffusion PSA_top, with tumour
Diffusion converged, nt = 7, ext_test = 2.307950e-07, prostate_test =
7.578925e-04
```


PSA, t = 7, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

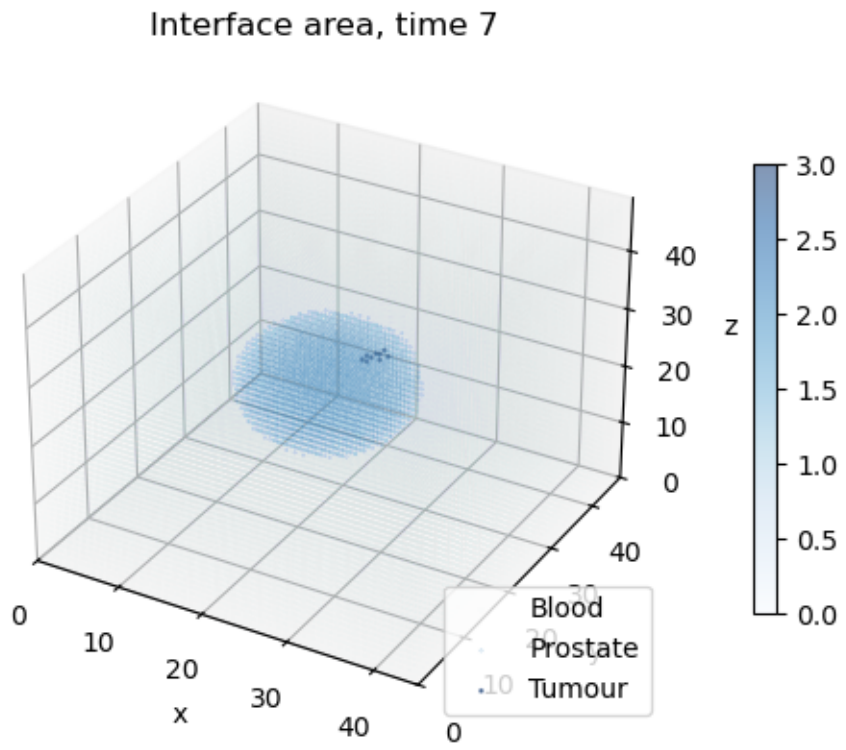

PSA, t = 7, plane at k = 28
PSA, t = 7, plane at j = 27
PSA, t = 7, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

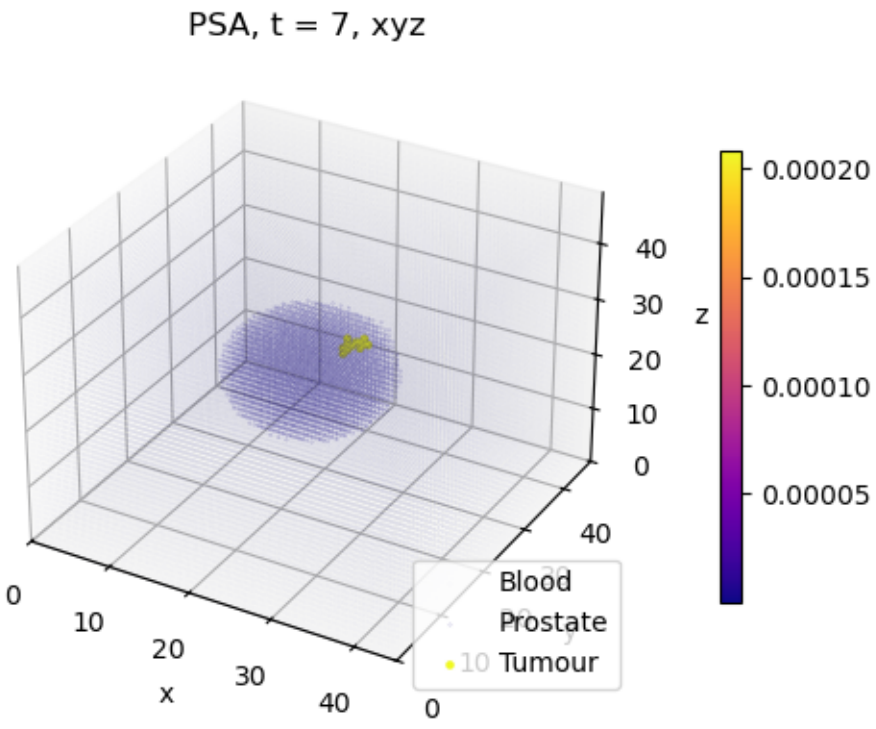

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

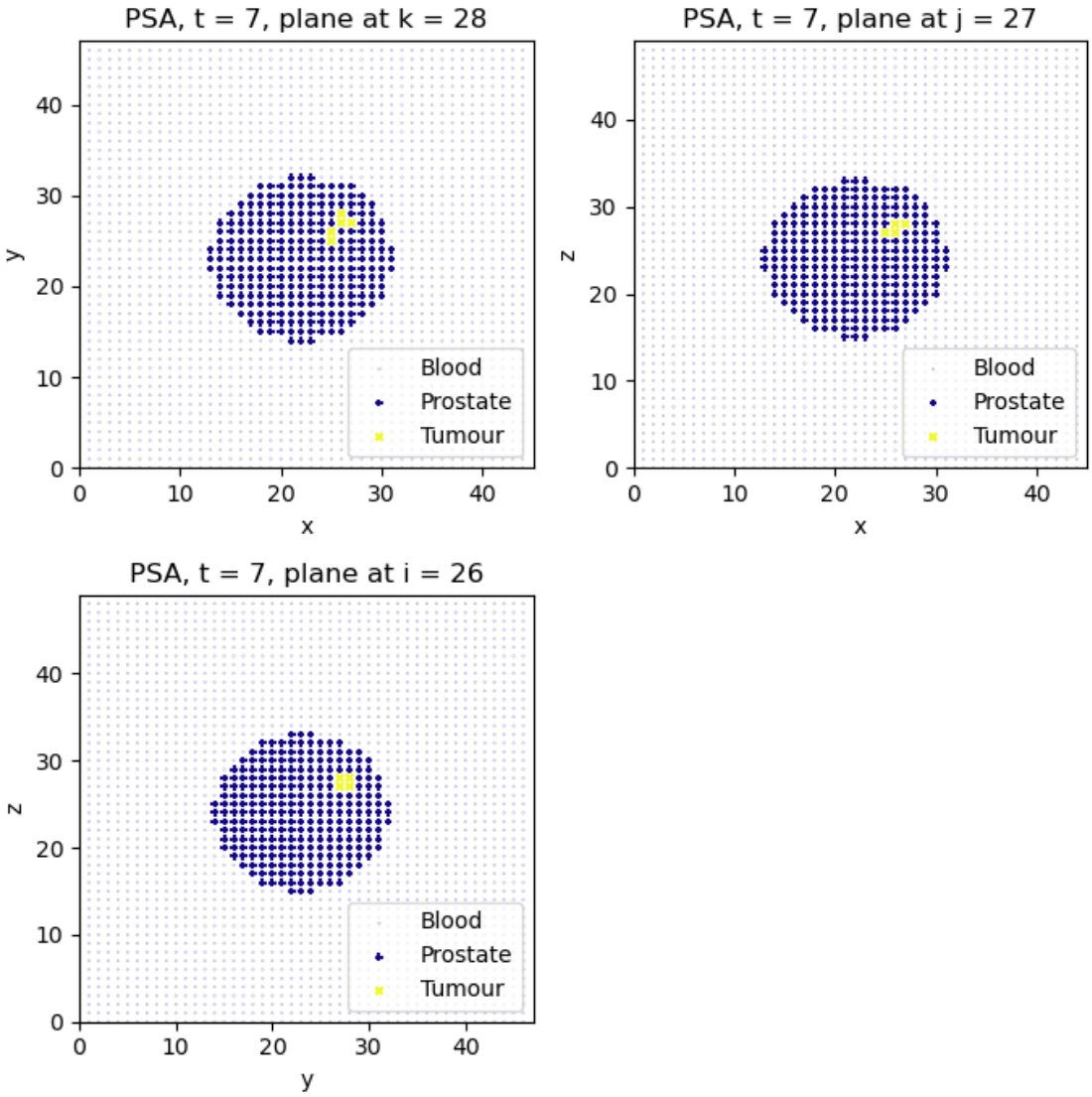

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

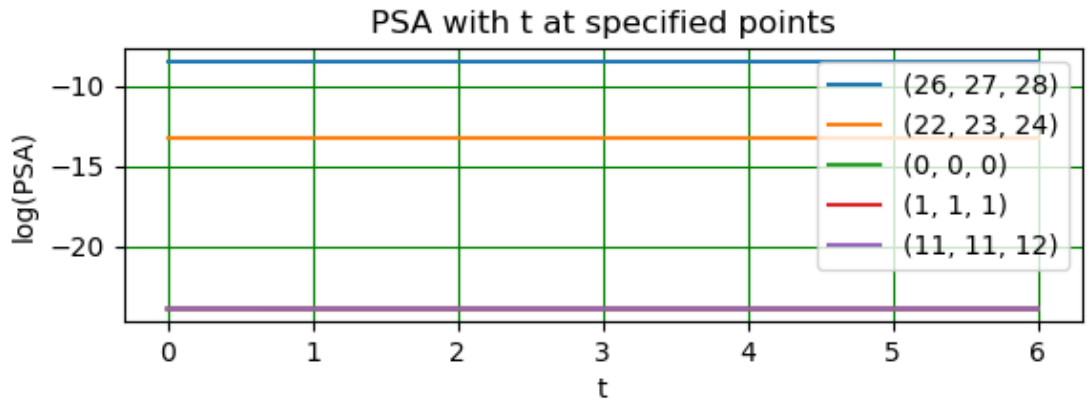

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

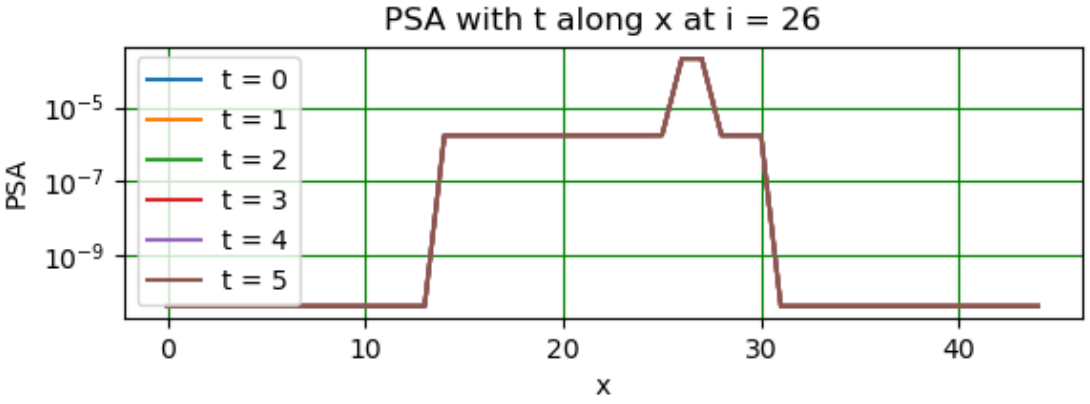


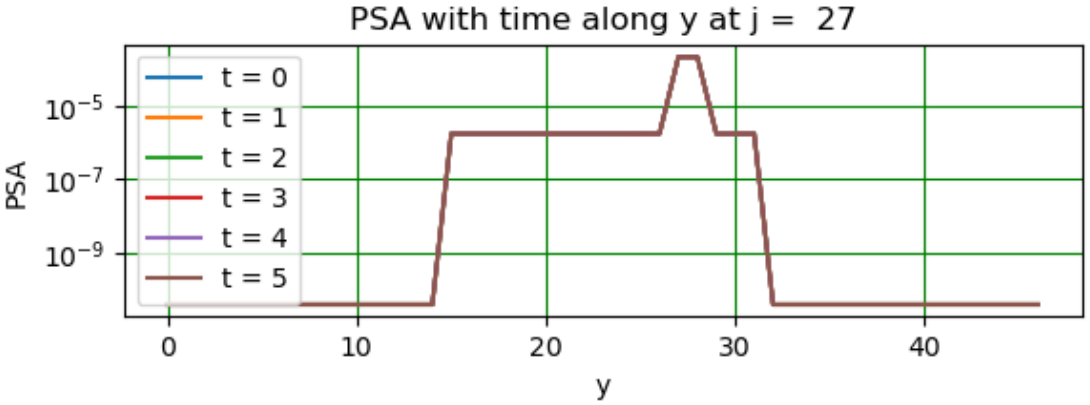

```
At time 7:
No. tumour cells 9, prostate cells 4138, blood cells 99488
PSA_top_level 4.083541e-11
------------------------------------------------------------------------------
--------------------------
Time 8
Diffusion PSA_top, with tumour
Diffusion converged, nt = 7, ext_test = 3.109455e-07, prostate_test =
6.065445e-04
```


PSA, t = 7, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

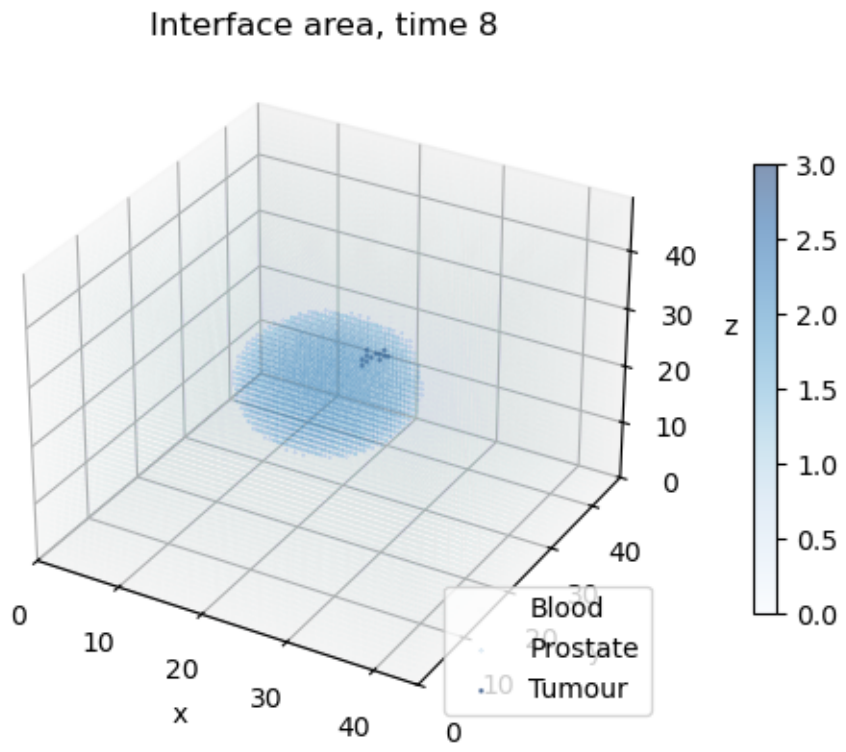

PSA, t = 7, plane at k = 28
PSA, t = 7, plane at j = 27
PSA, t = 7, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
6
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

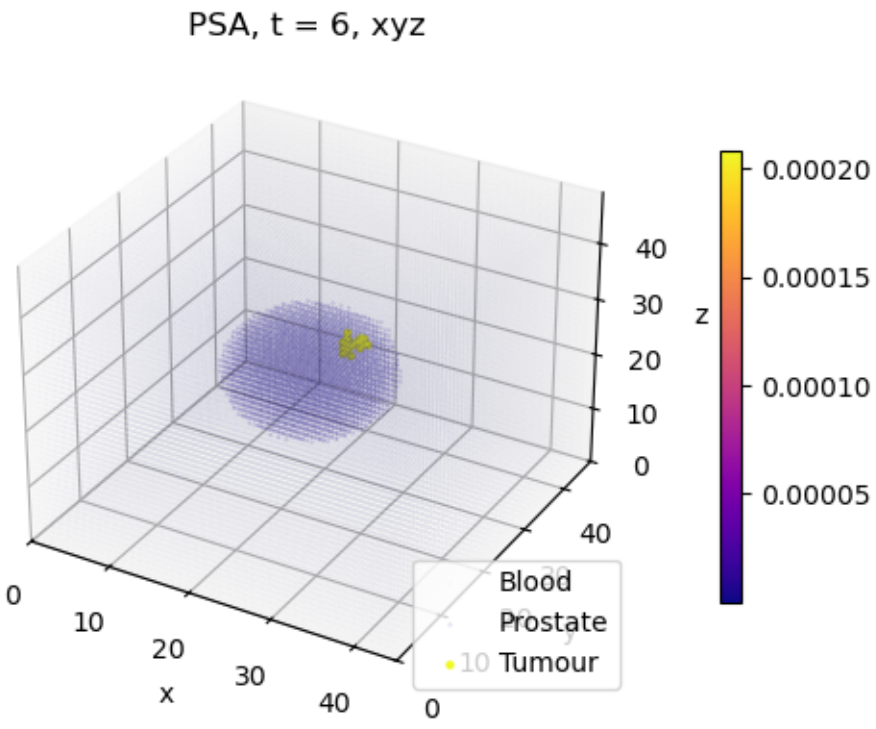

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

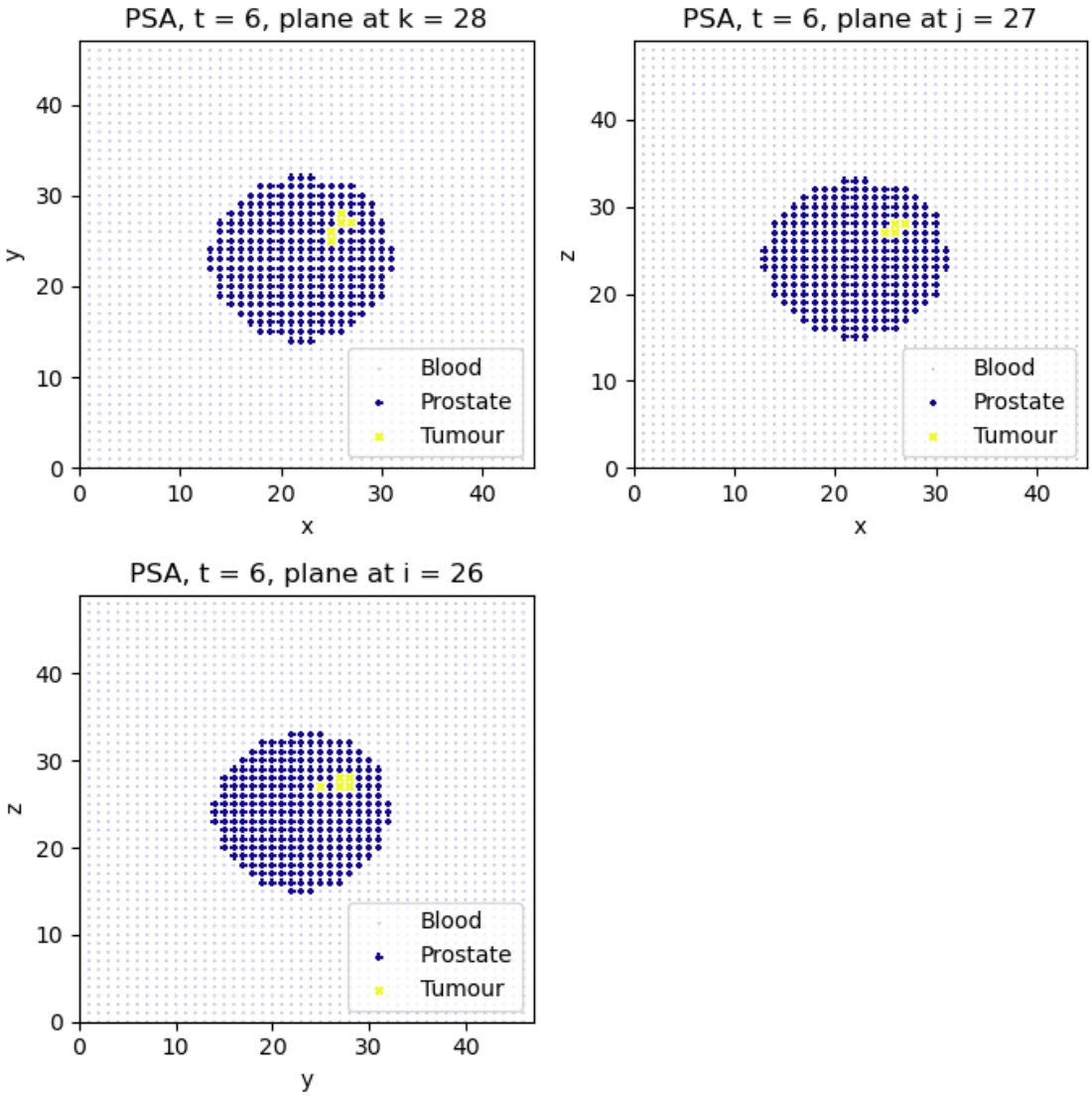

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4
t = 5

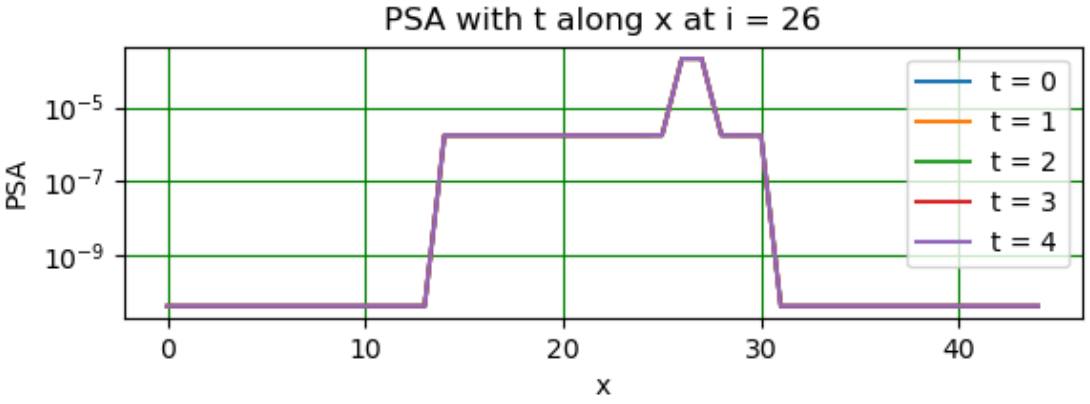


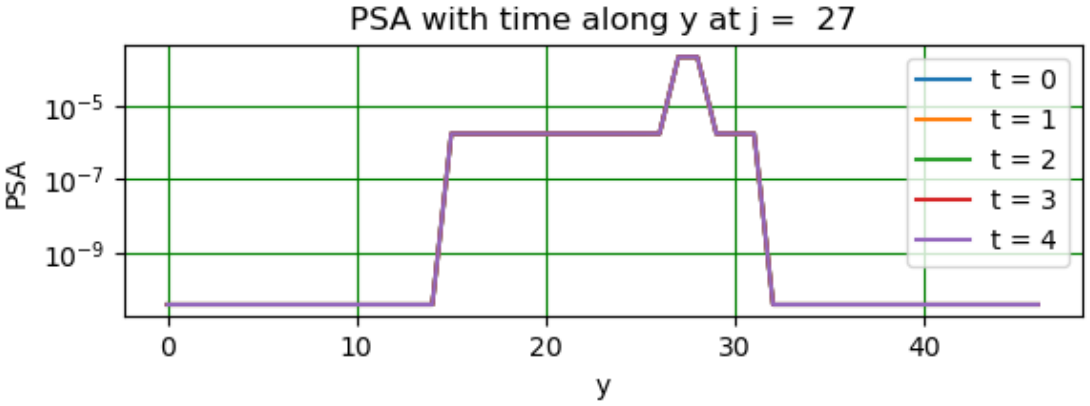

```
At time 8:
No. tumour cells 12, prostate cells 4138, blood cells 99485
PSA_top_level 4.076282e-11
------------------------------------------------------------------------------
--------------------------
Time 9
Diffusion PSA_top, with tumour
Diffusion converged, nt = 6, ext_test = 8.849160e-09, prostate_test =
8.657586e-04
```

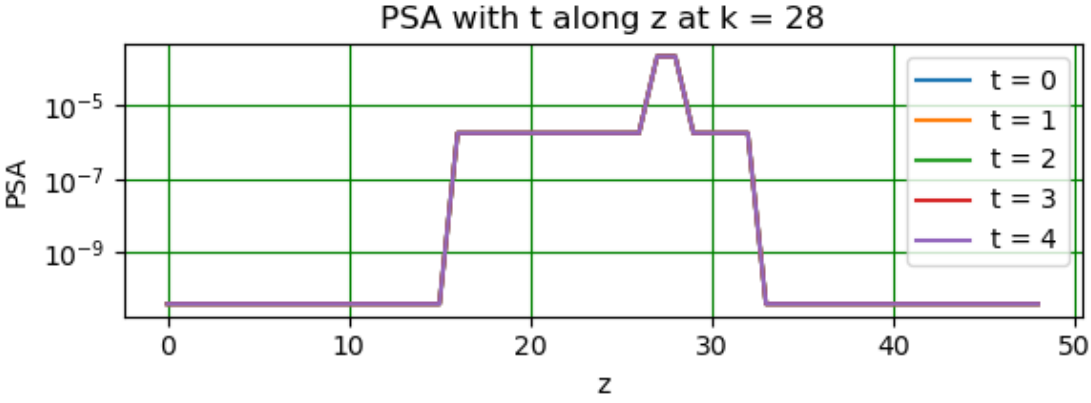

PSA, t = 6, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

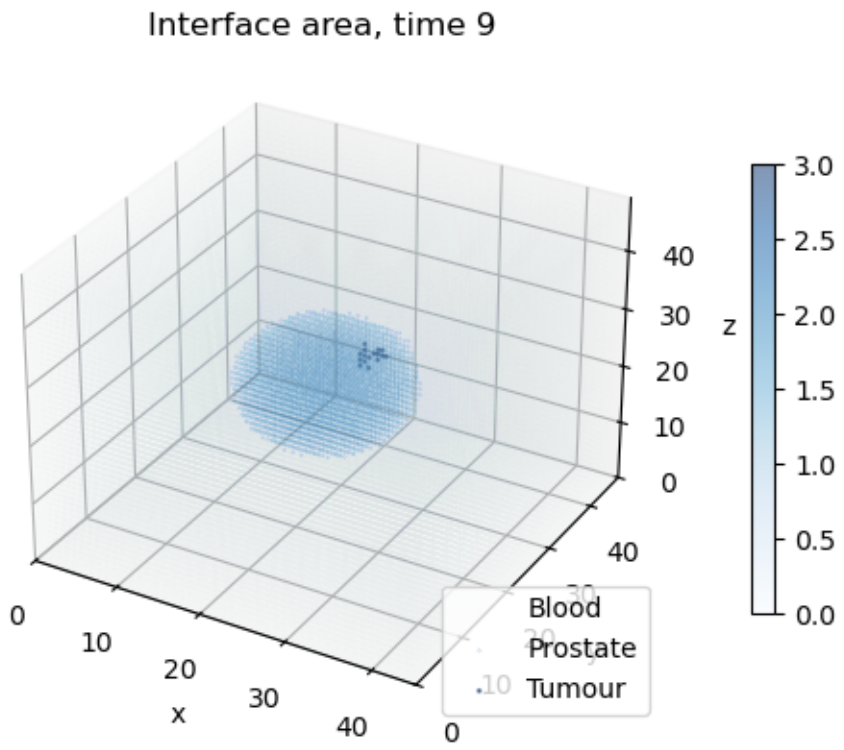

PSA, t = 6, plane at k = 28
PSA, t = 6, plane at j = 27
PSA, t = 6, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

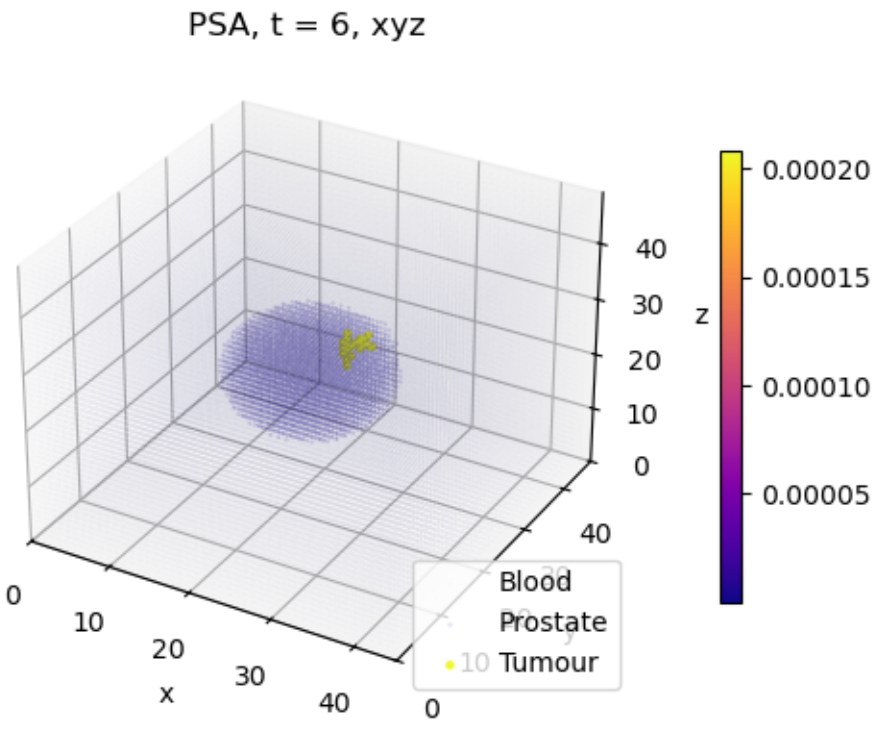

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4

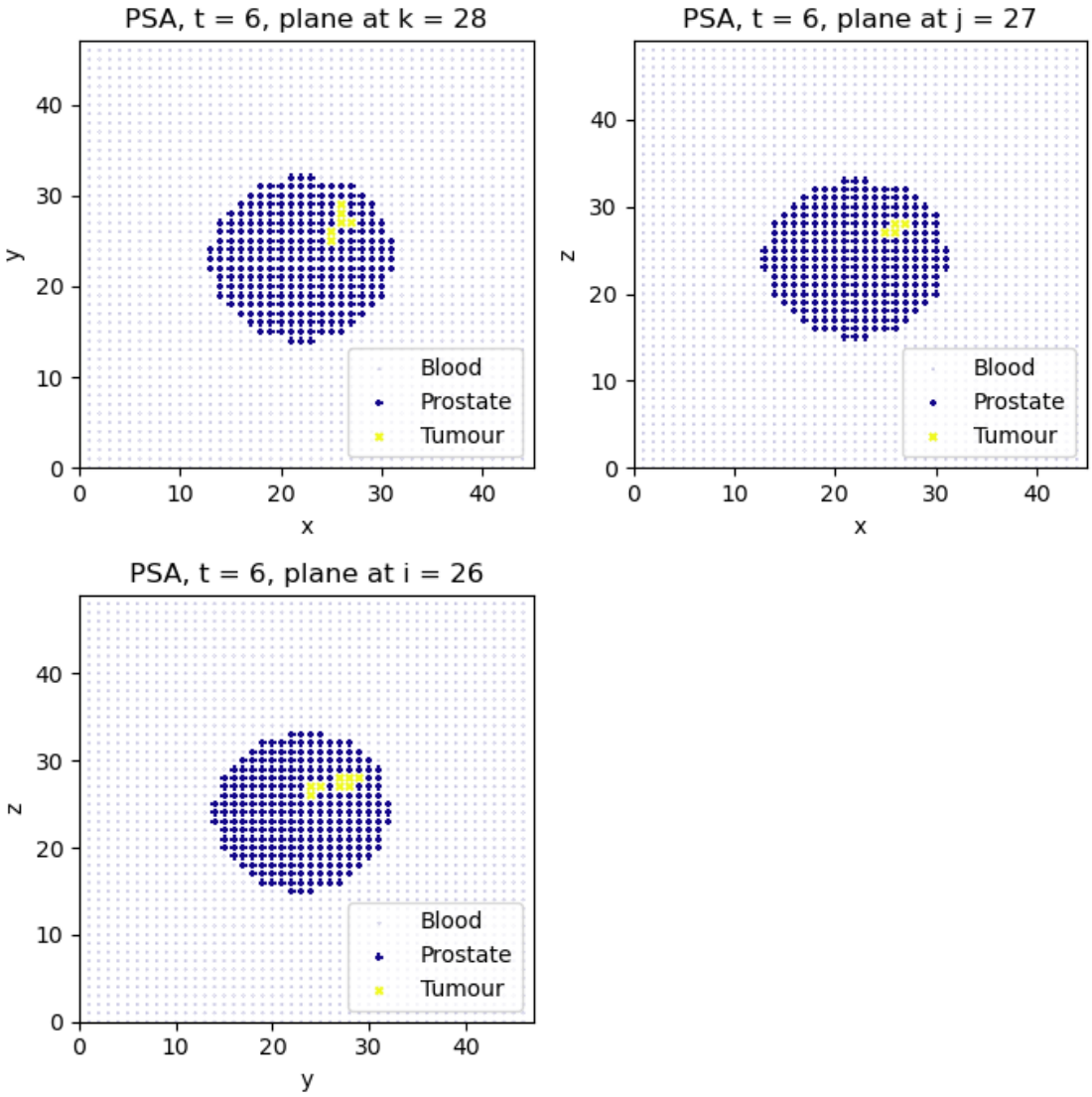

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4

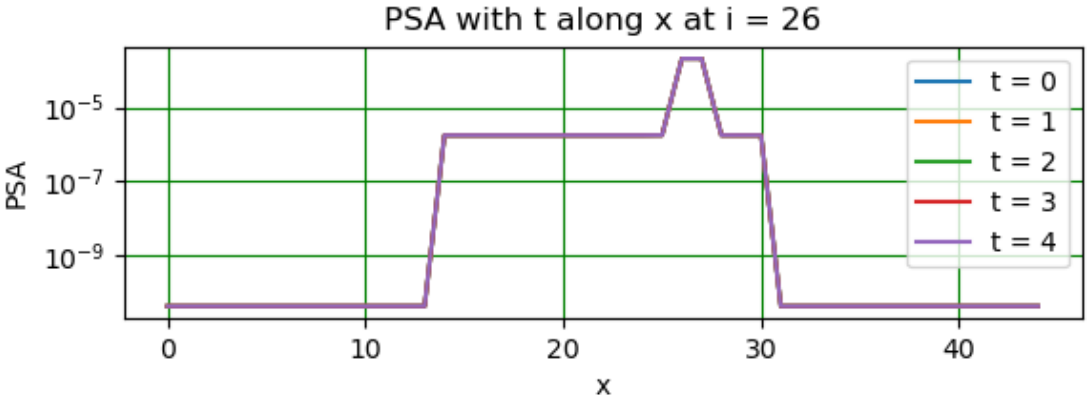


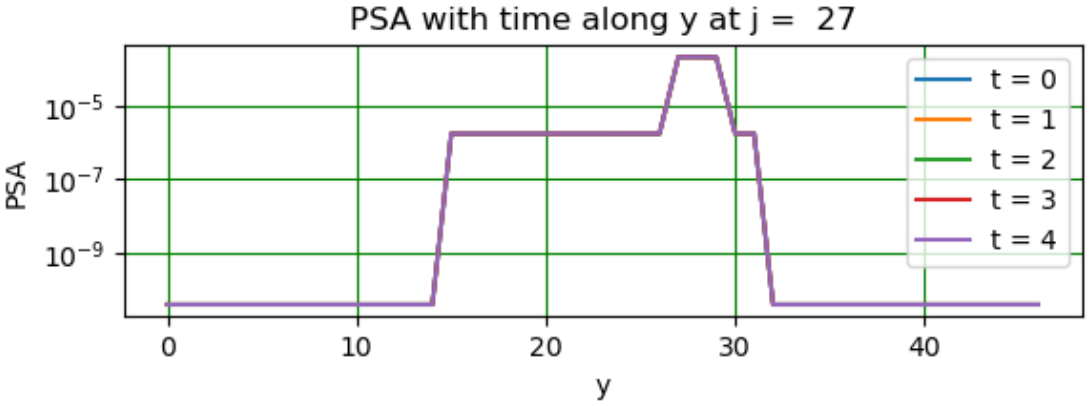

```
At time 9:
No. tumour cells 16, prostate cells 4138, blood cells 99481
PSA_top_level 4.076282e-11
-------------------------------------------------------------------------------
--------------------------
Time 10
Diffusion PSA_top, with tumour
Diffusion converged, nt = 6, ext_test = 7.158181e-07, prostate_test =
6.632289e-04
```

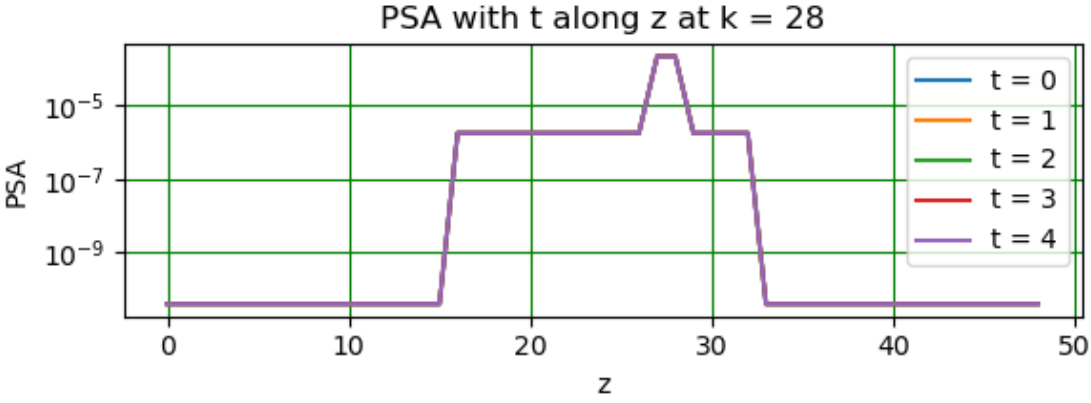

PSA, t = 6, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

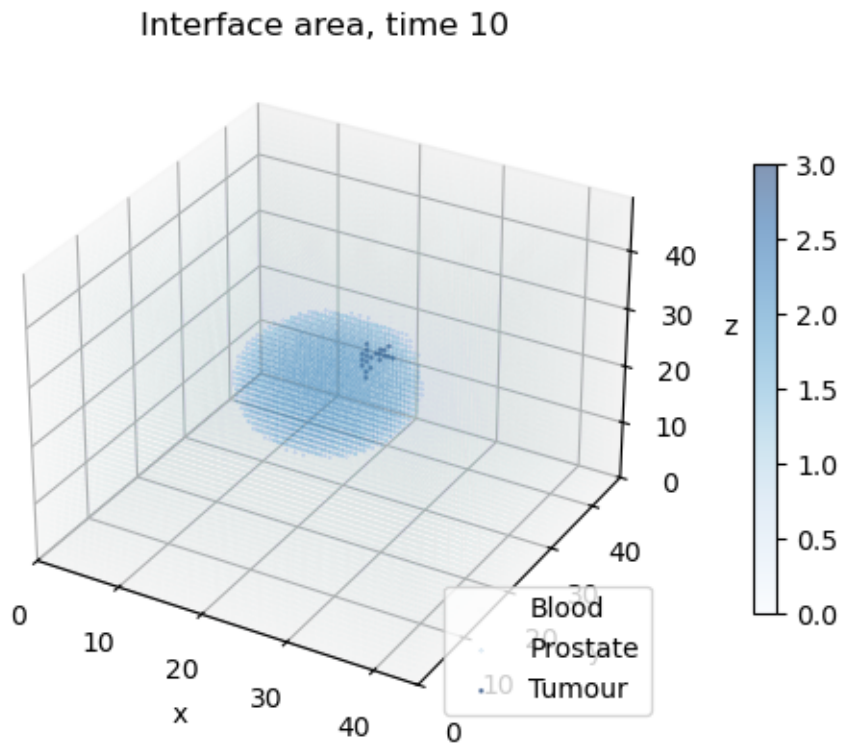

PSA, t = 6, plane at k = 28
PSA, t = 6, plane at j = 27
PSA, t = 6, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0
1
2
3
4
5
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

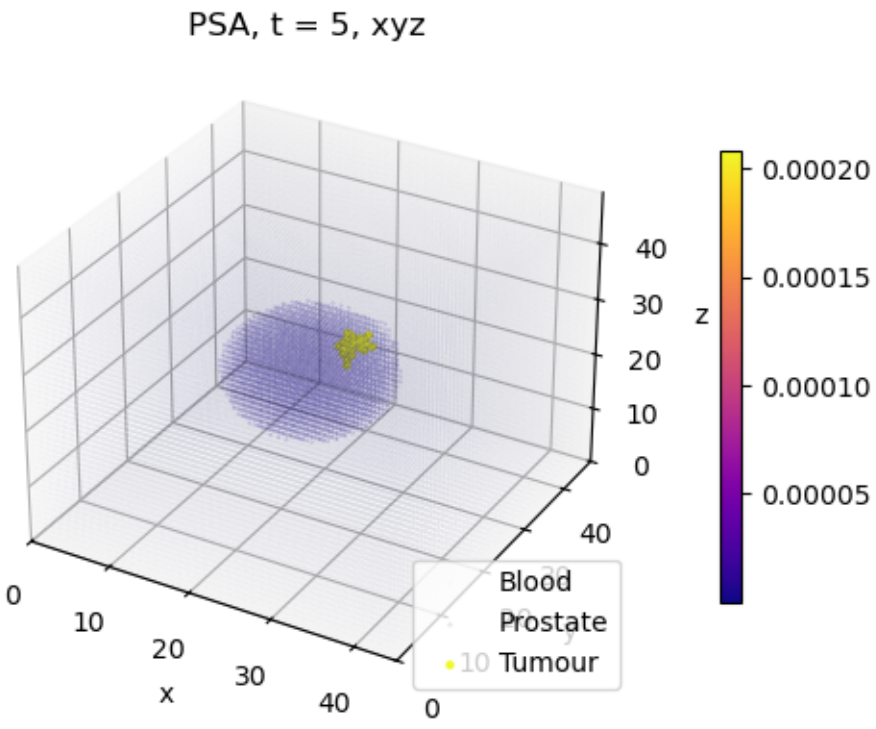

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3
t = 4

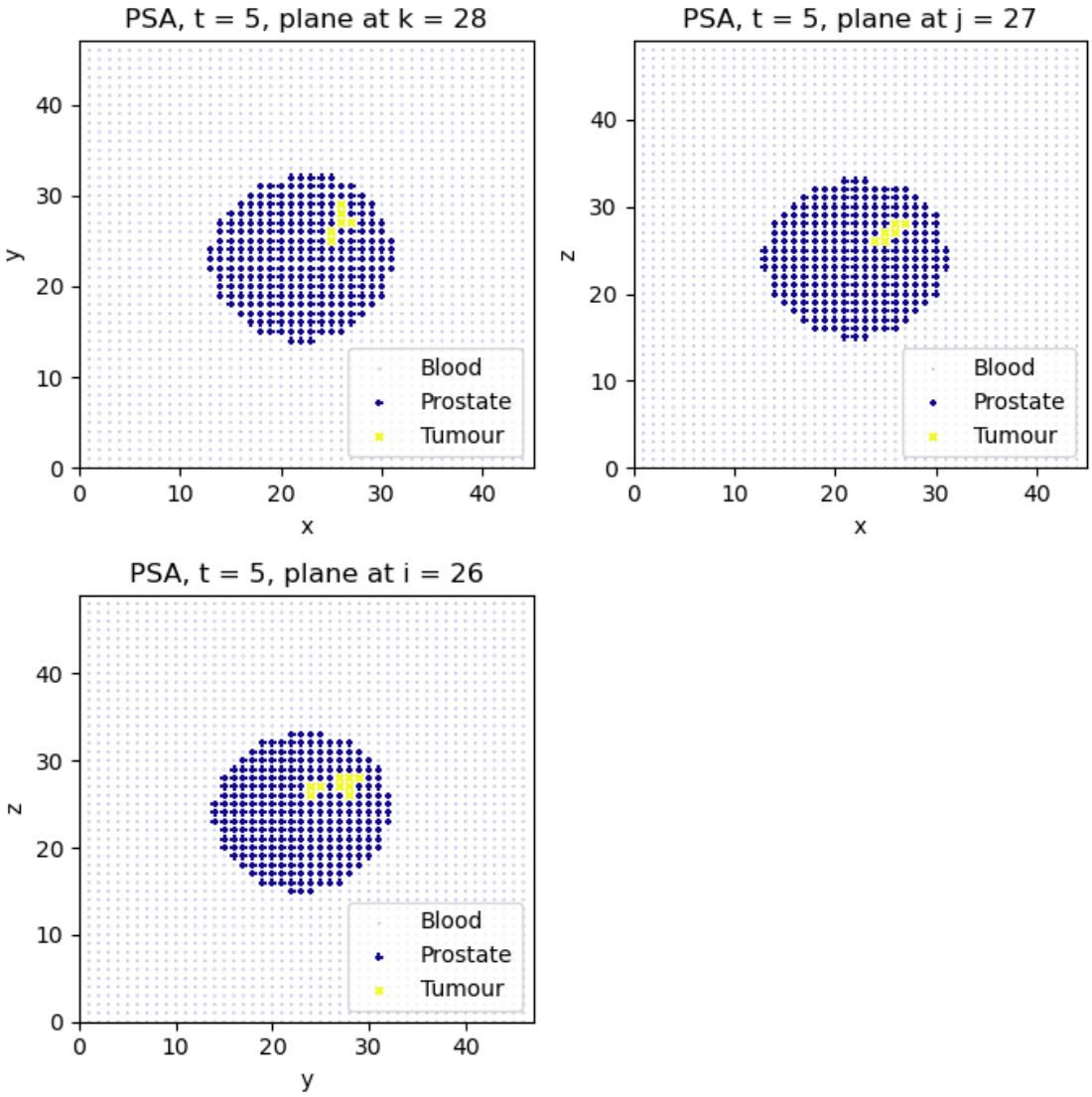

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3
t = 4

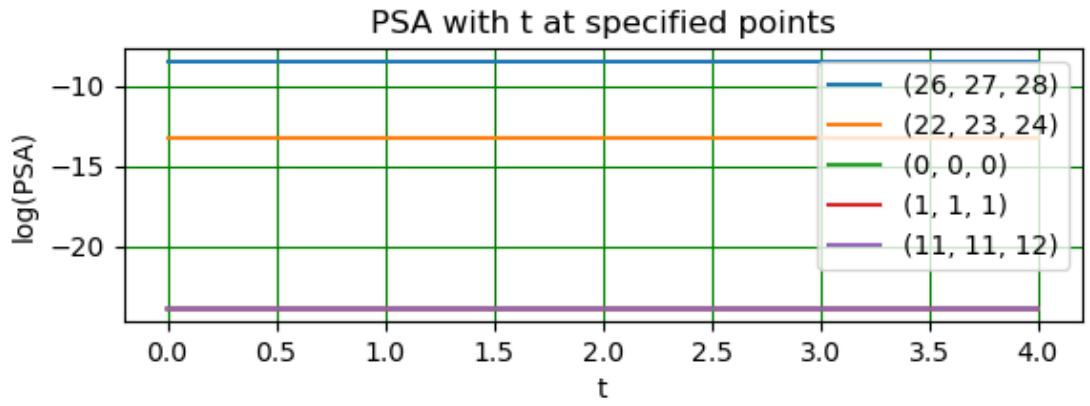

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3
t = 4

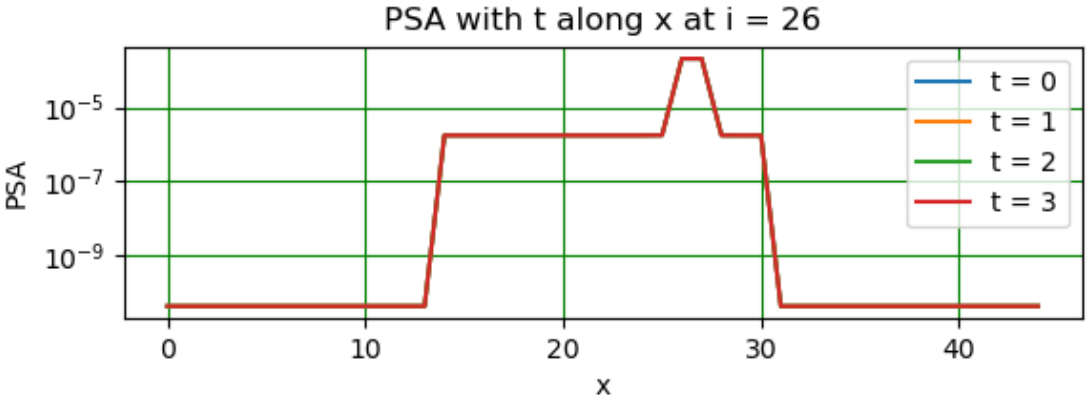


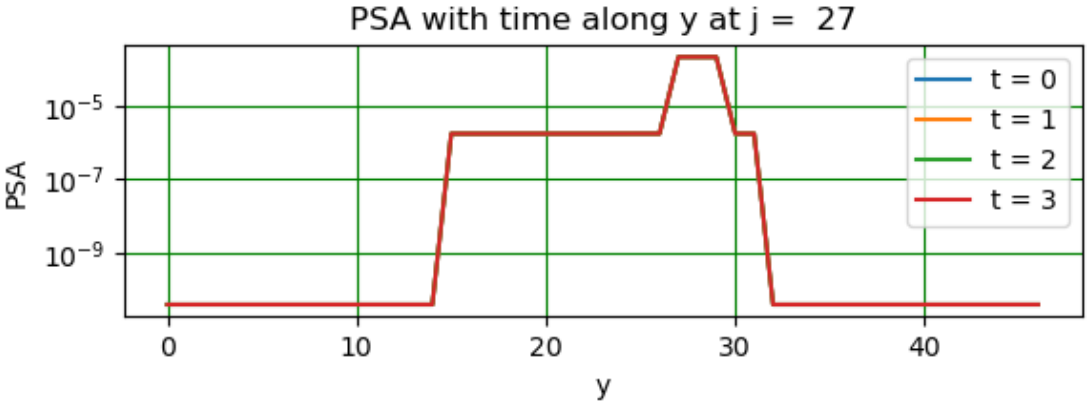

```
At time 10:
No. tumour cells 20, prostate cells 4138, blood cells 99477
PSA_top_level 4.081562e-11
------------------------------------------------------------------------------
--------------------------
Time 11
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 1.303618e-07, prostate_test =
9.953439e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

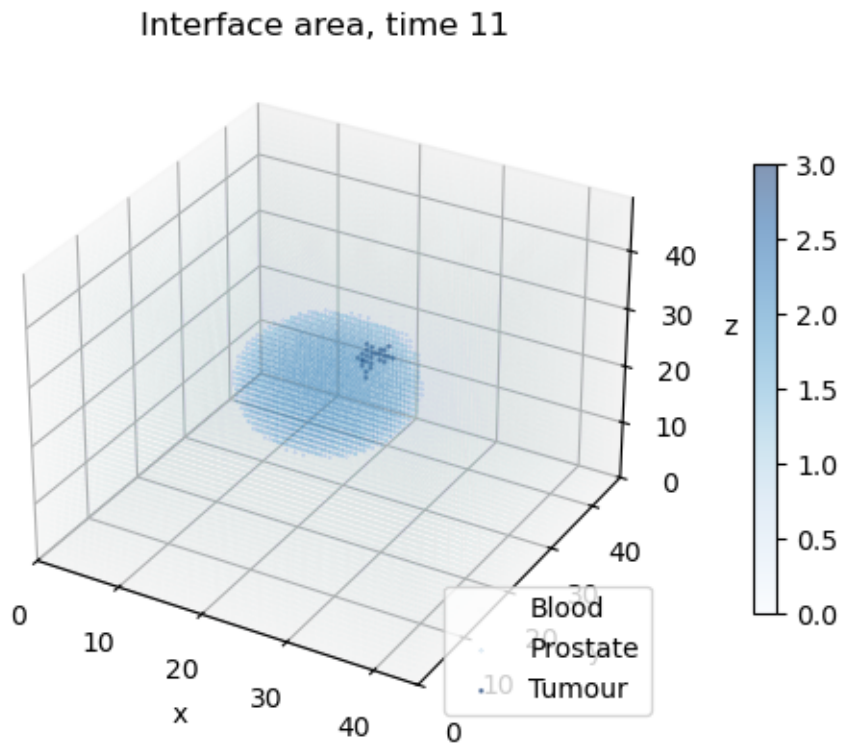

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

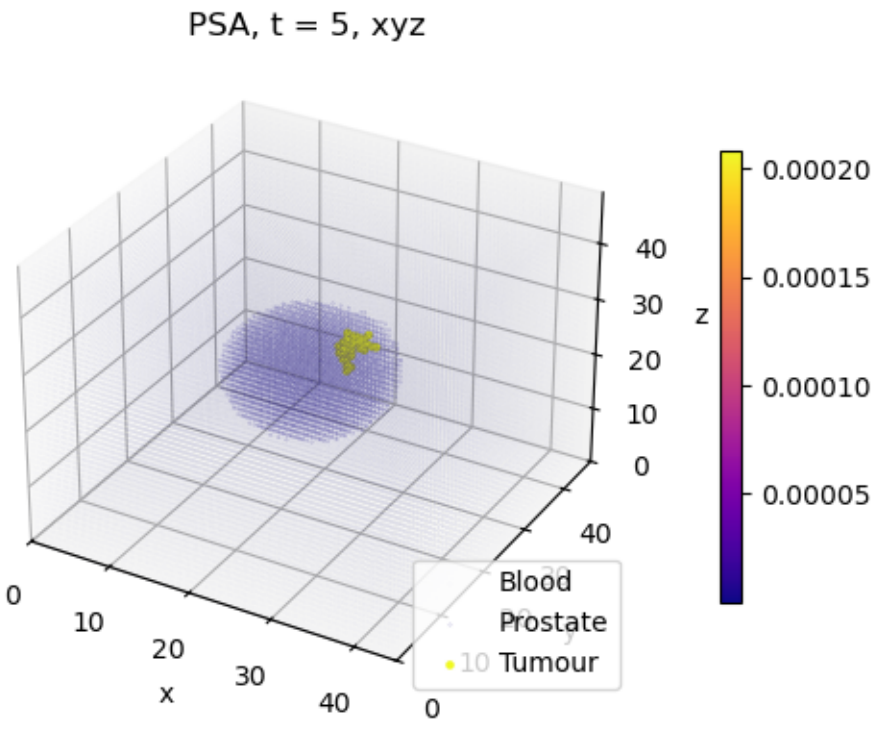

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

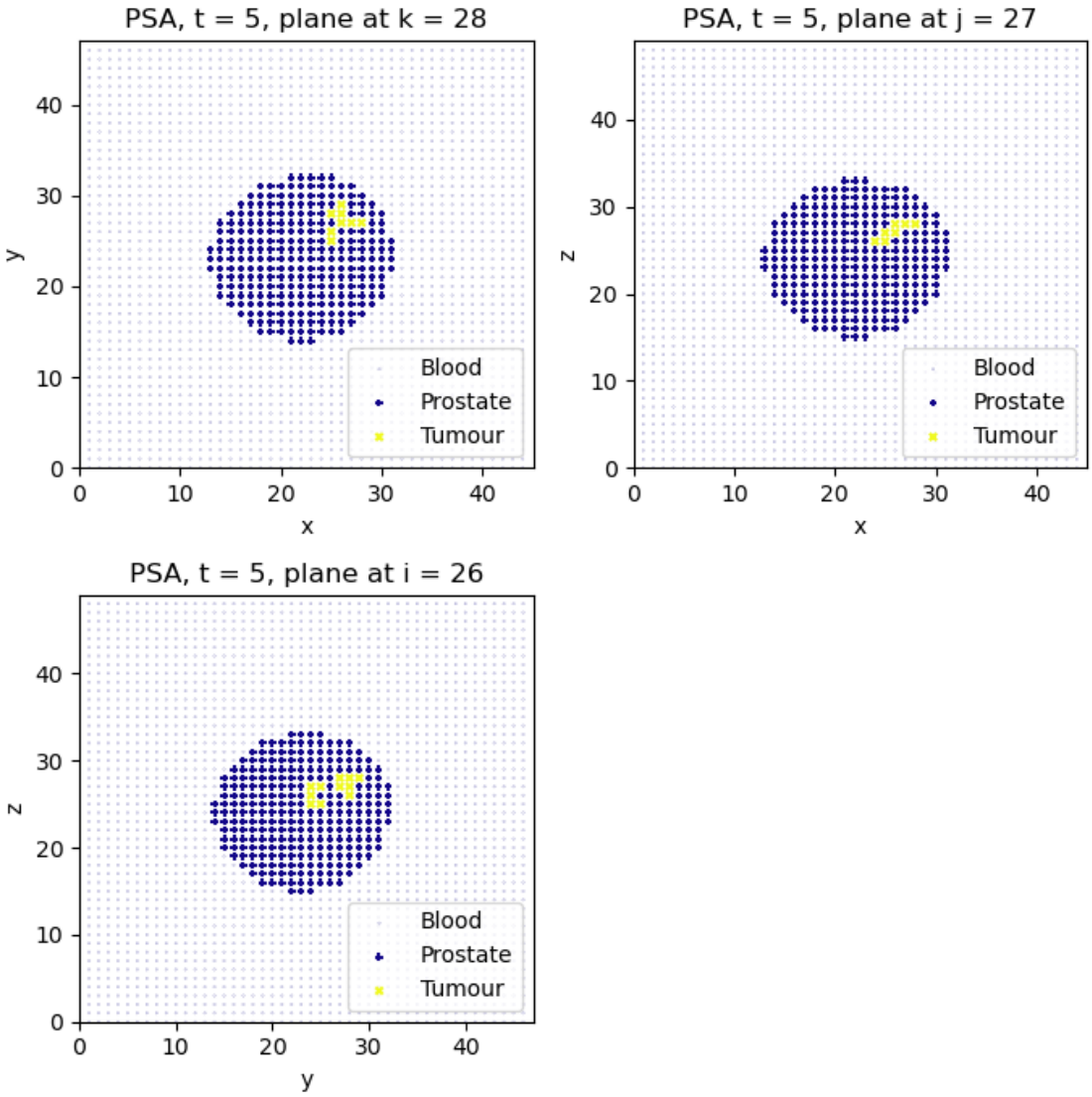

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3

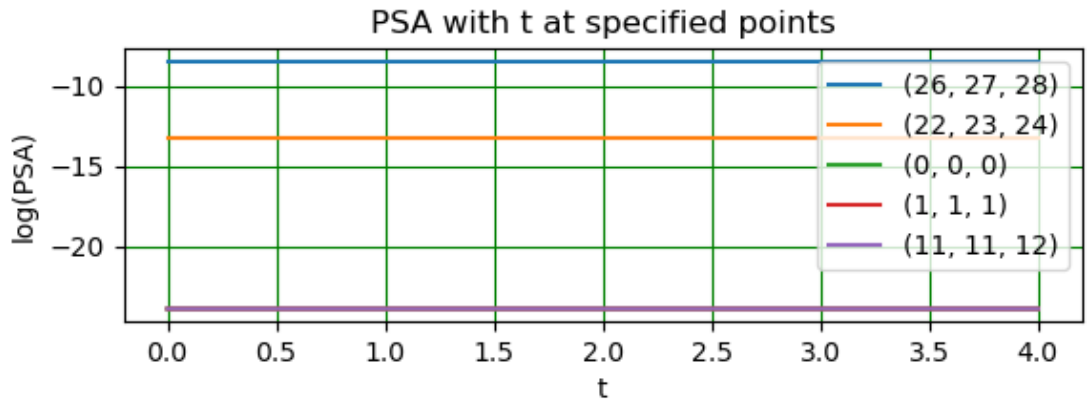

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

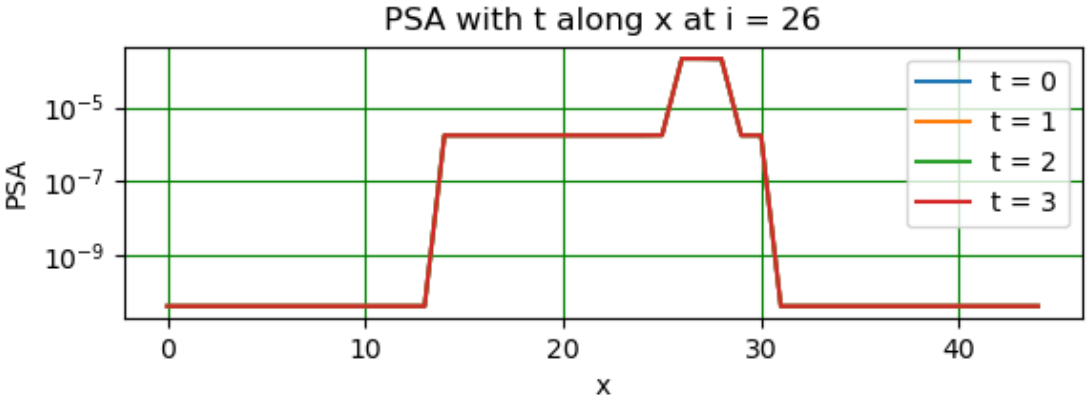


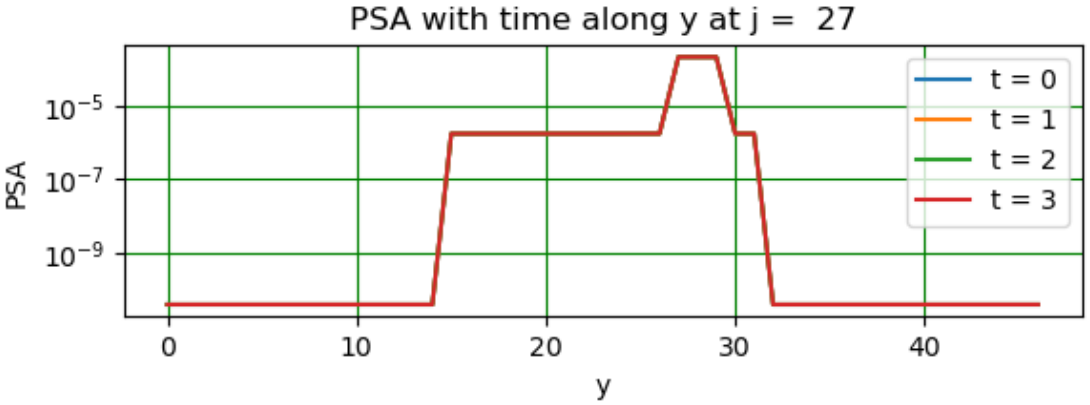

```
At time 11:
No. tumour cells 26, prostate cells 4138, blood cells 99471
PSA_top_level 4.082390e-11
------------------------------------------------------------------------------
--------------------------
Time 12
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 1.749223e-06, prostate_test =
7.769934e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

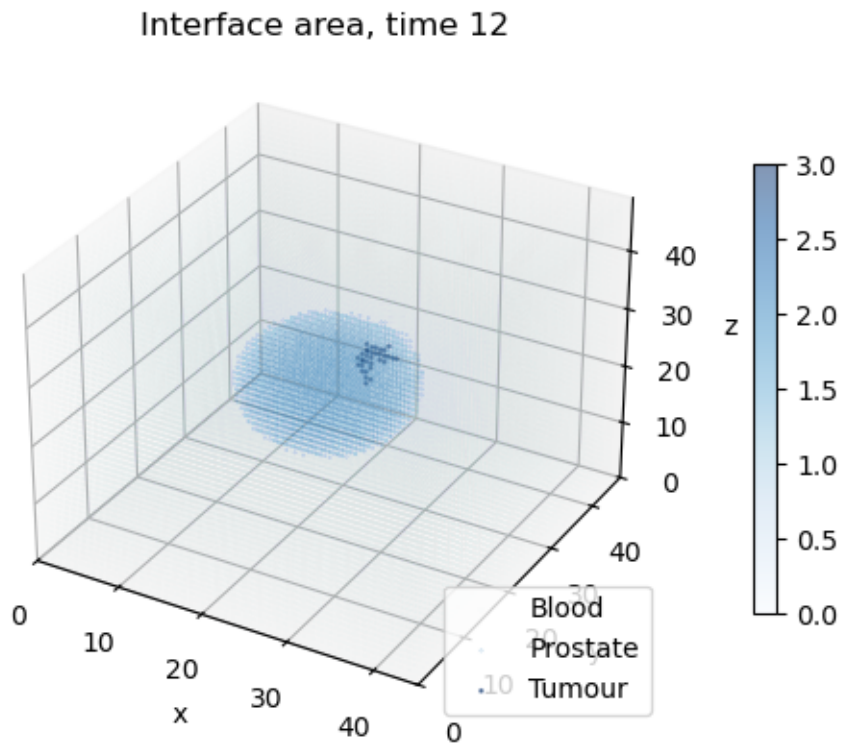

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

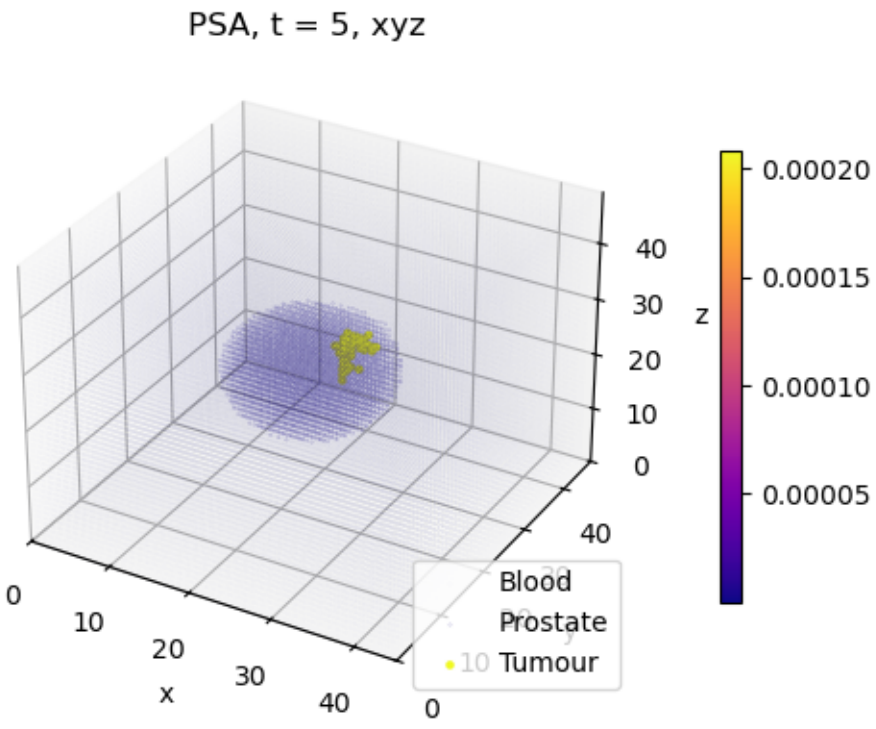

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

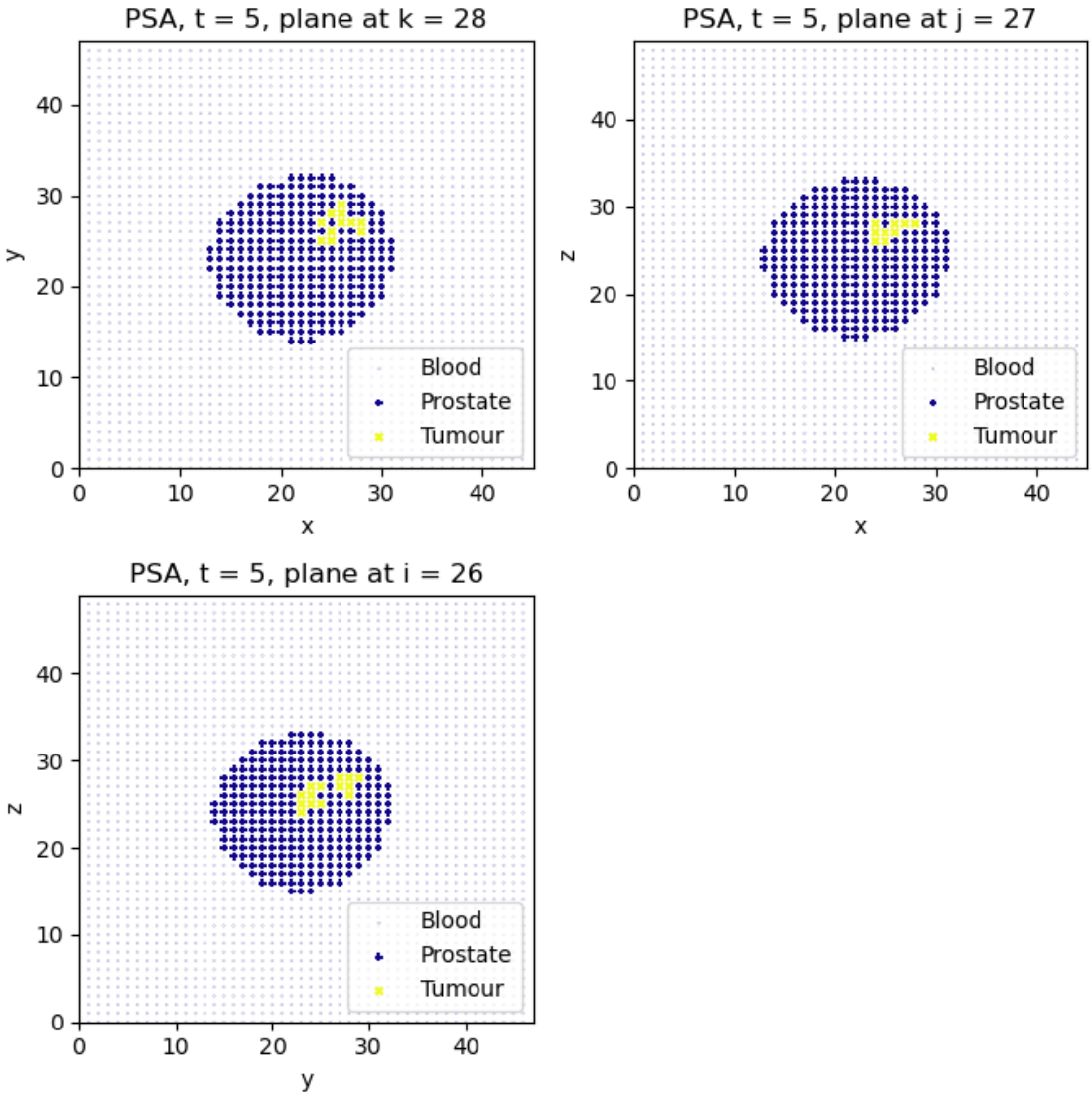

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3

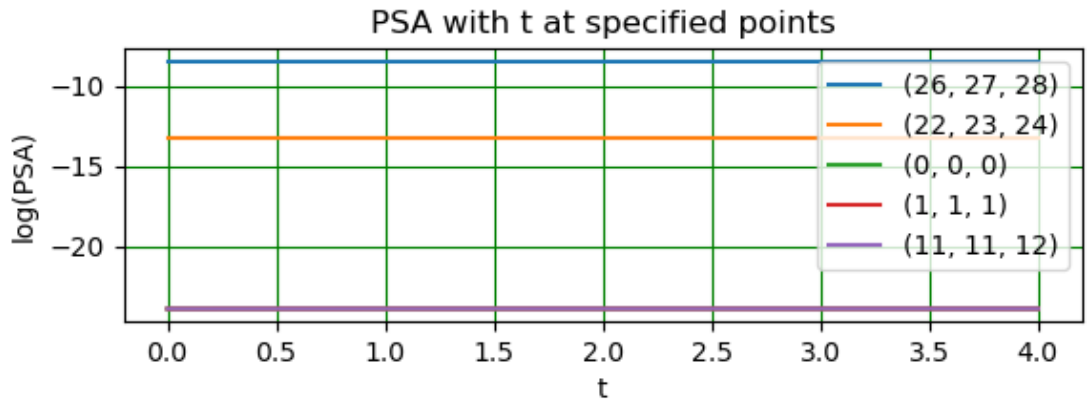

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

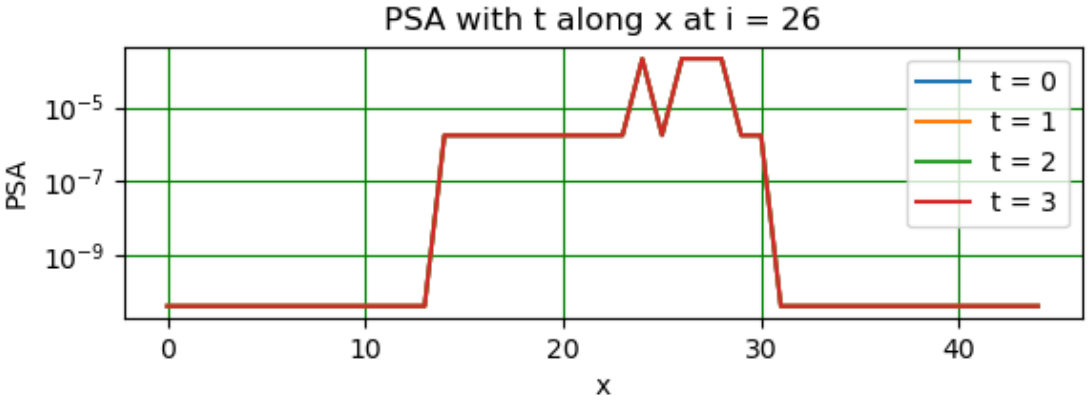


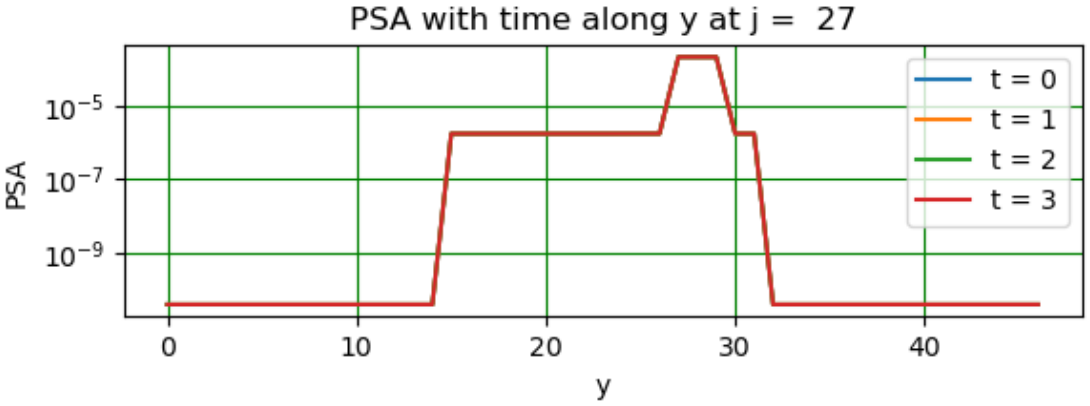

```
At time 12:
No. tumour cells 33, prostate cells 4138, blood cells 99464
PSA_top_level 4.088473e-11
------------------------------------------------------------------------------
--------------------------
Time 13
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 1.001171e-05, prostate_test =
6.173869e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

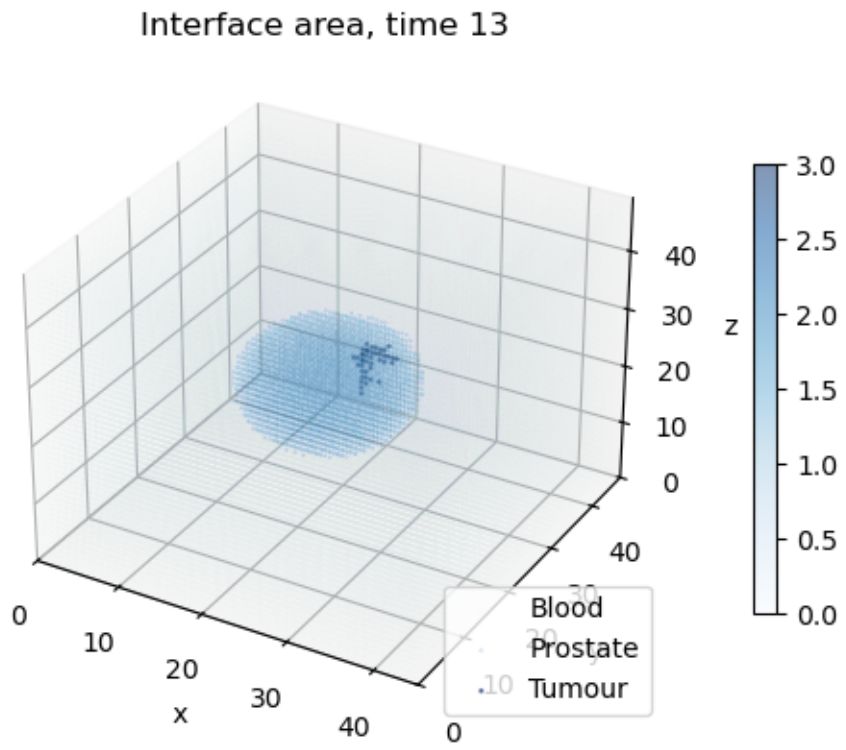

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

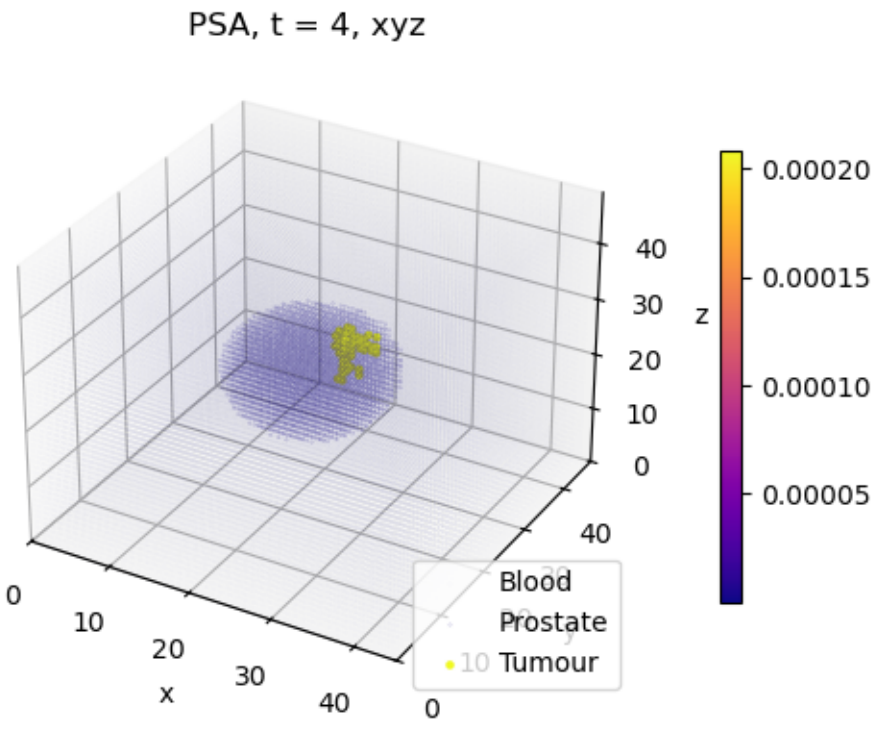

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

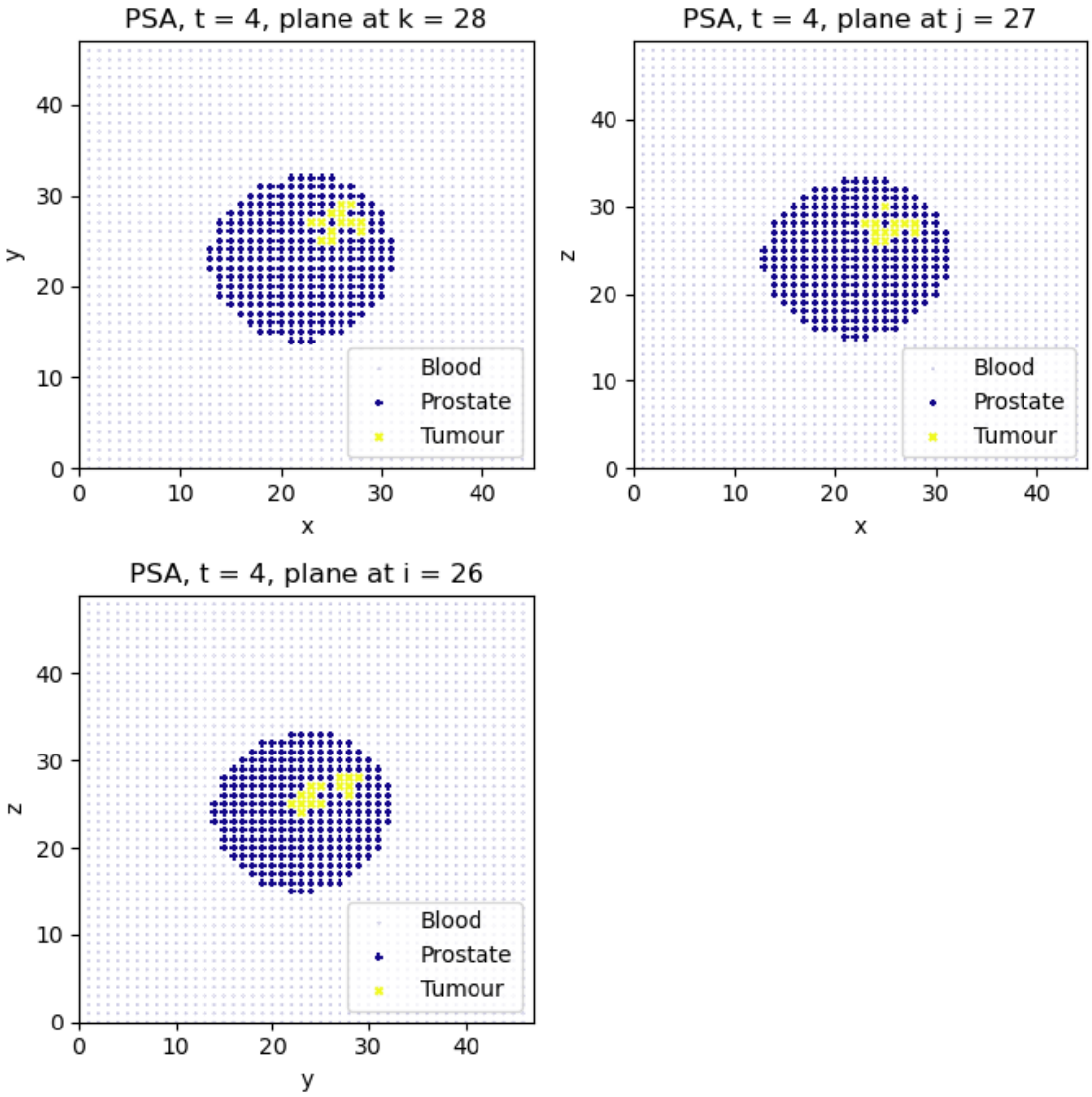

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3

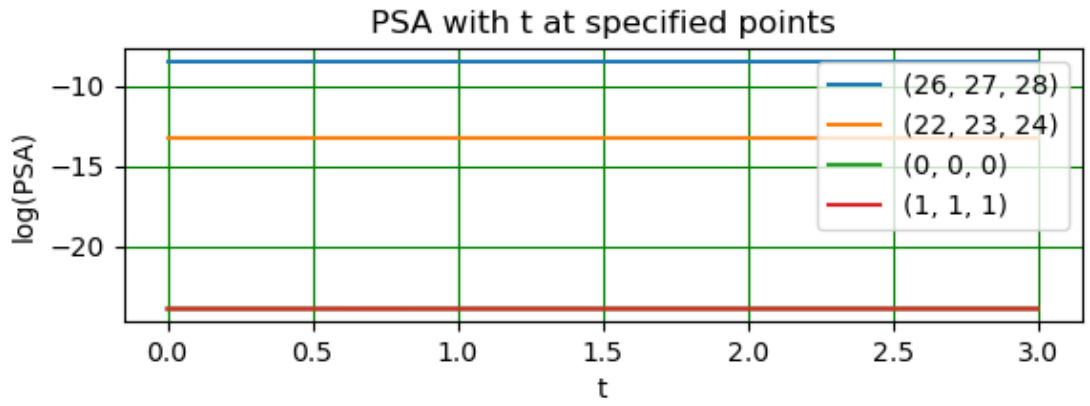

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

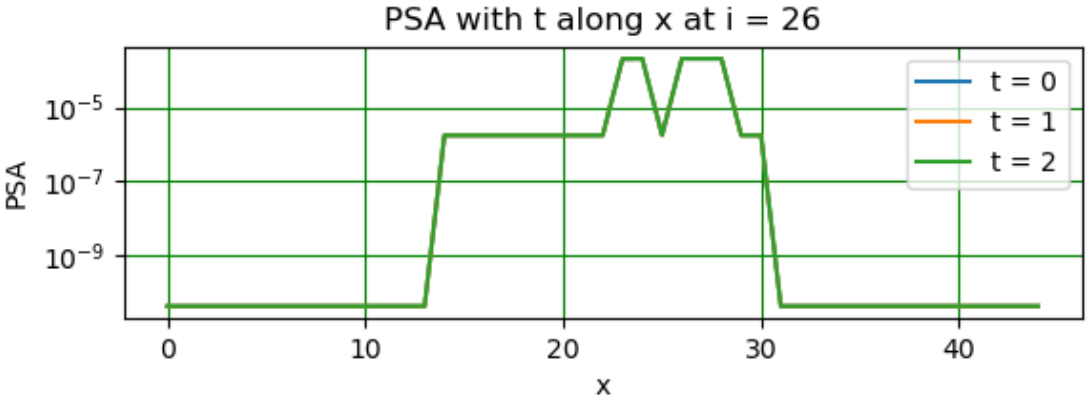


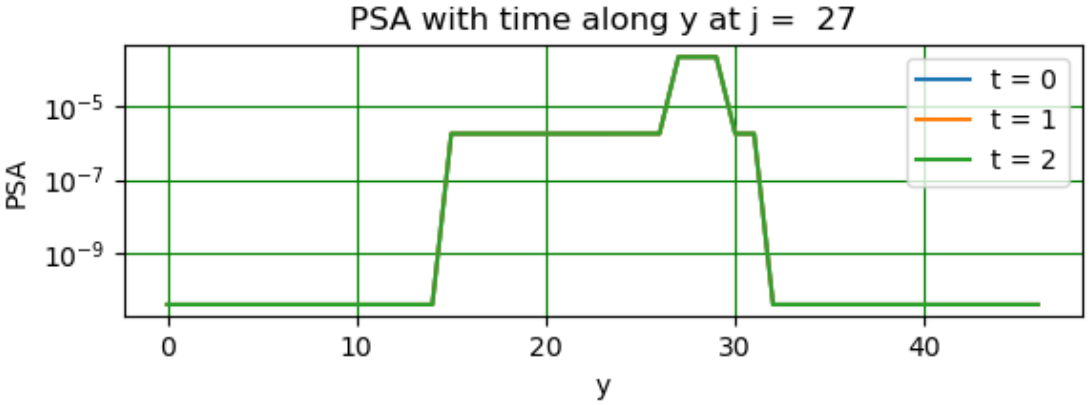

```
At time 13:
No. tumour cells 42, prostate cells 4138, blood cells 99455
PSA_top_level 4.111447e-11
-------------------------------------------------------------------------------
--------------------------
Time 14
Diffusion PSA_top, with tumour
Diffusion converged, nt = 4, ext_test = 2.170168e-05, prostate_test =
9.044889e-04
```

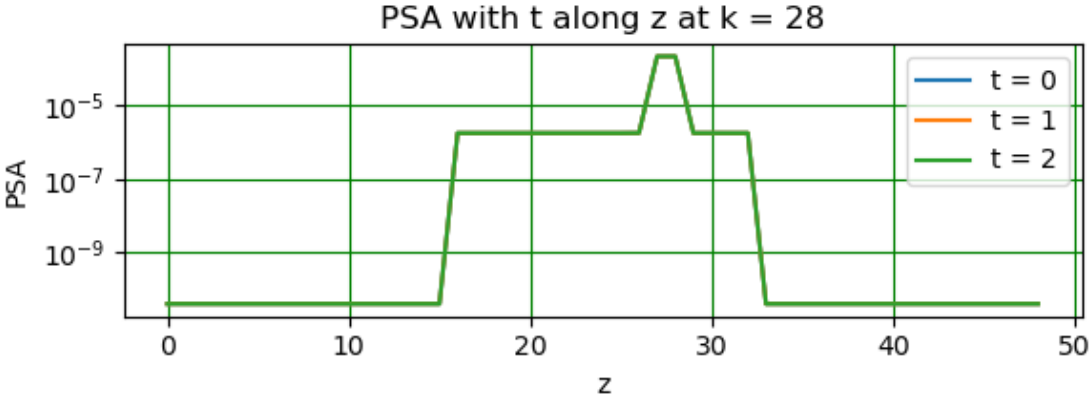

PSA, t = 4, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

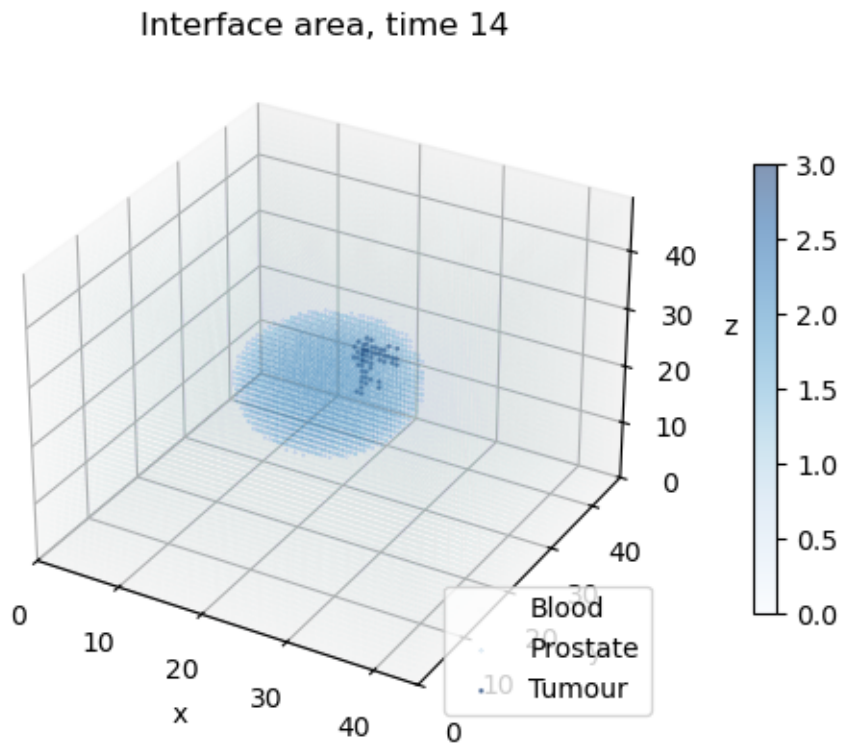

PSA, t = 4, plane at k = 28
PSA, t = 4, plane at j = 27
PSA, t = 4, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)

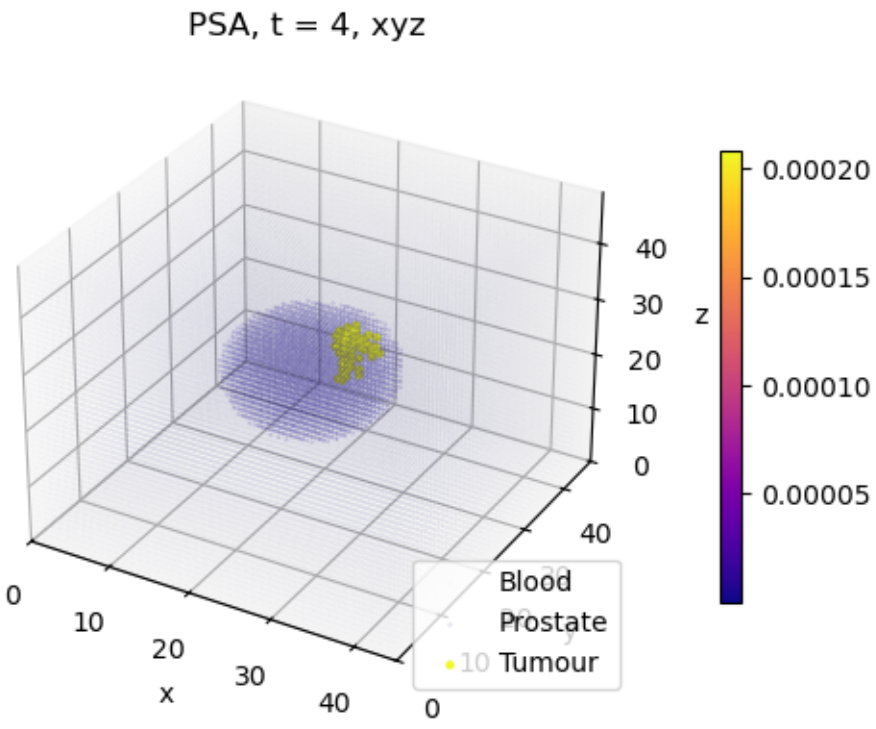

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2

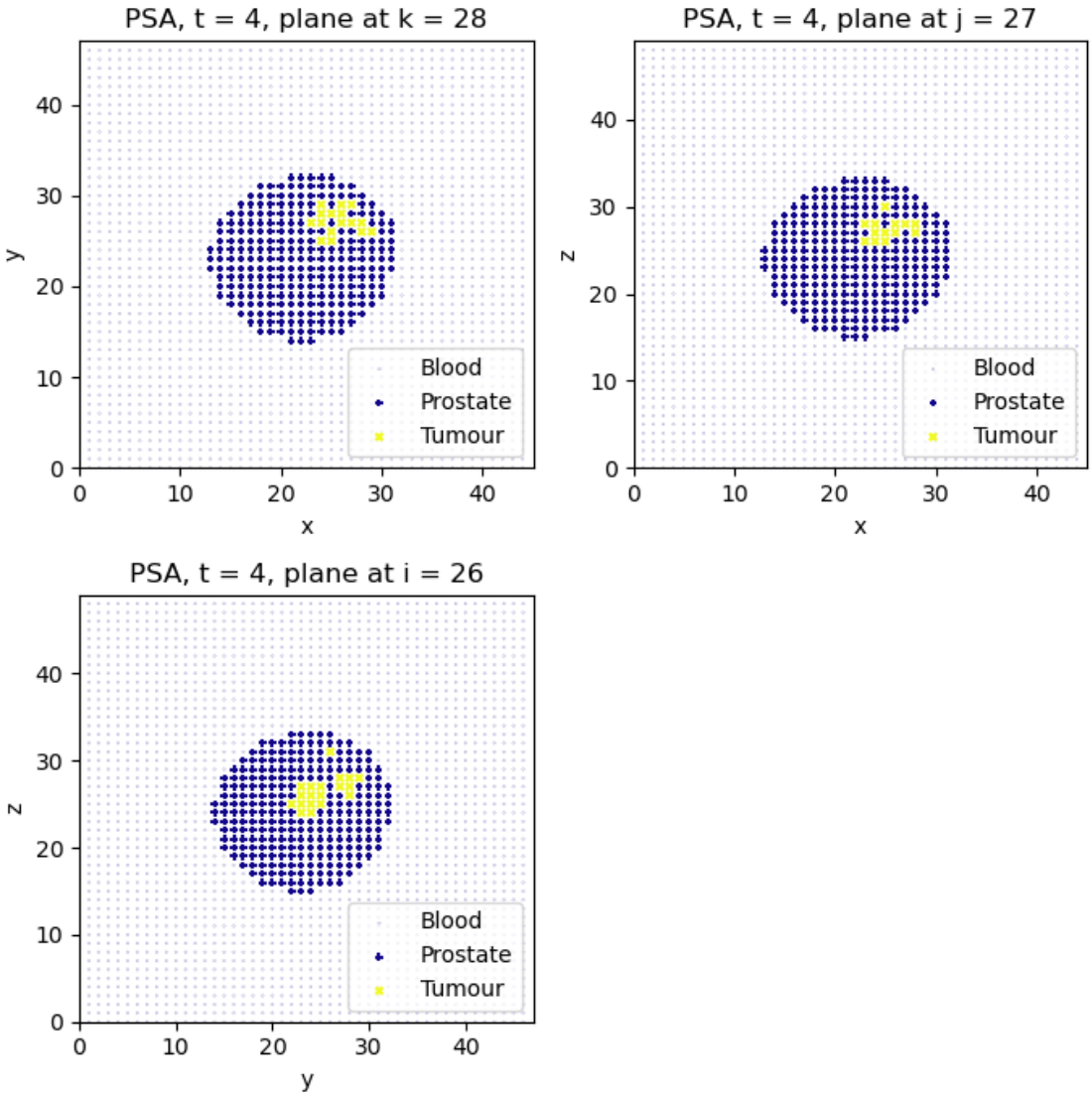

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2

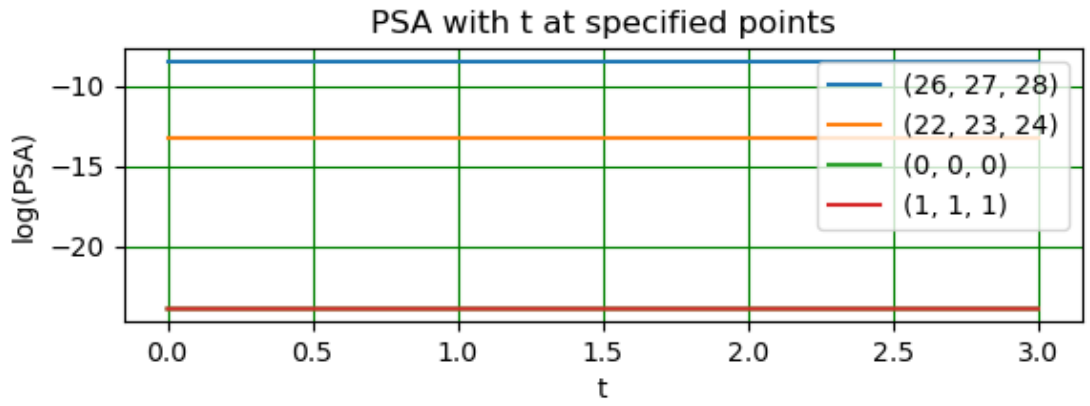

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2

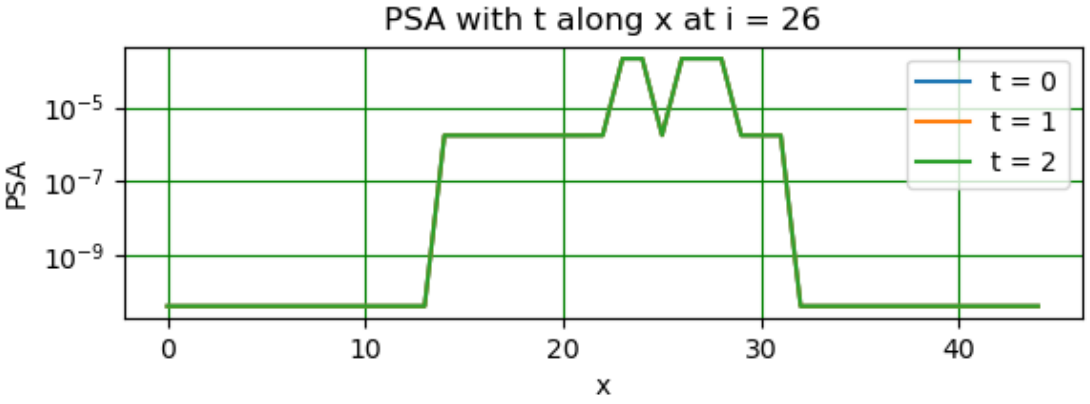


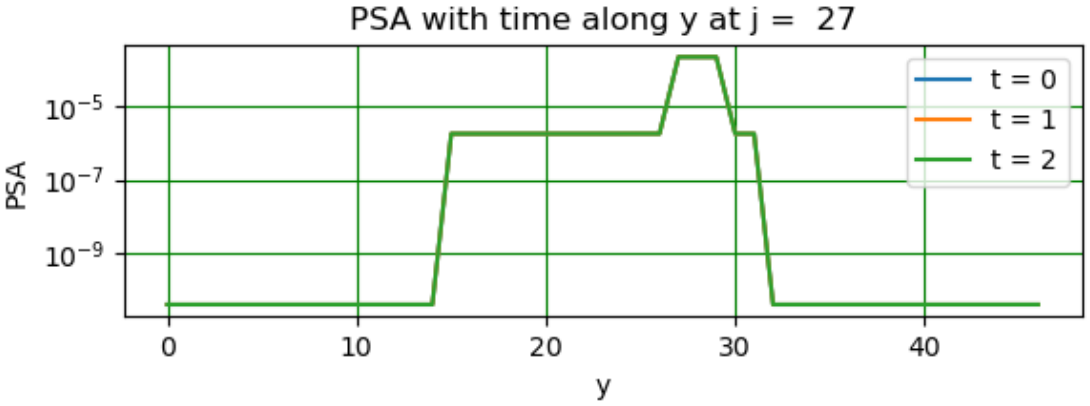

```
At time 14:
No. tumour cells 53, prostate cells 4138, blood cells 99444
PSA_top_level 4.126296e-11
------------------------------------------------------------------------------
--------------------------
Time 15
Diffusion PSA_top, with tumour
Diffusion converged, nt = 4, ext_test = 1.165480e-05, prostate_test =
7.254844e-04
```

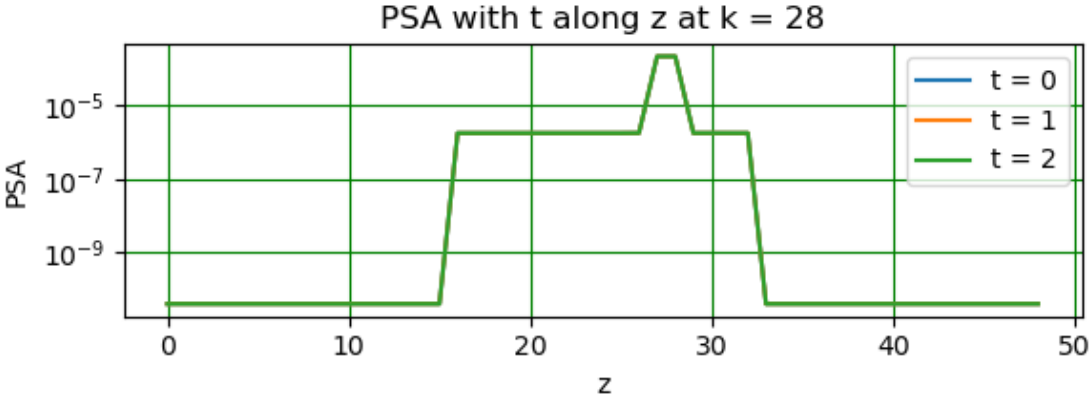

PSA, t = 4, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

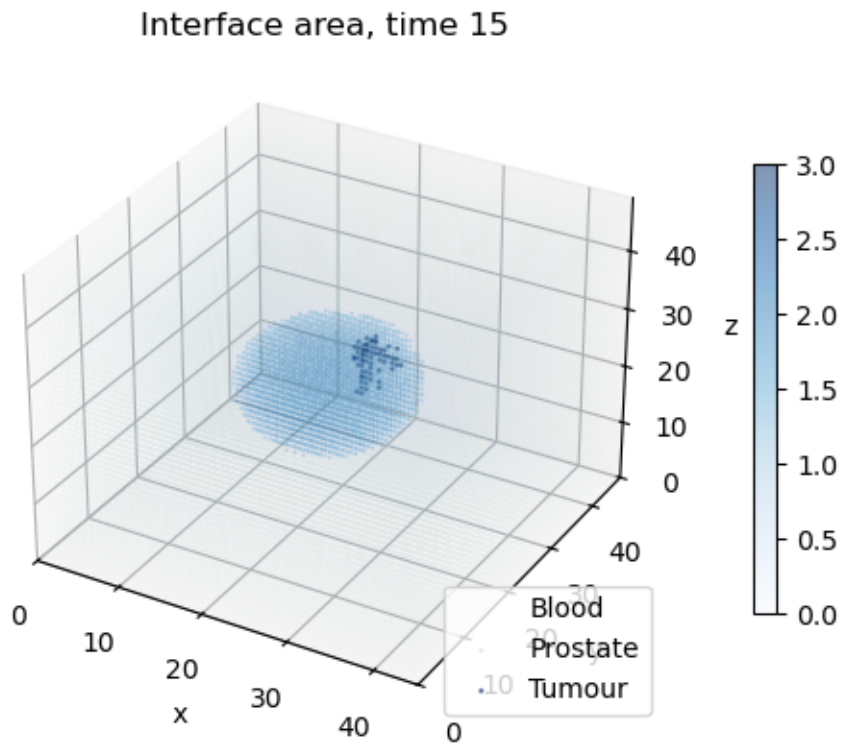

PSA, t = 4, plane at k = 28
PSA, t = 4, plane at j = 27
PSA, t = 4, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)

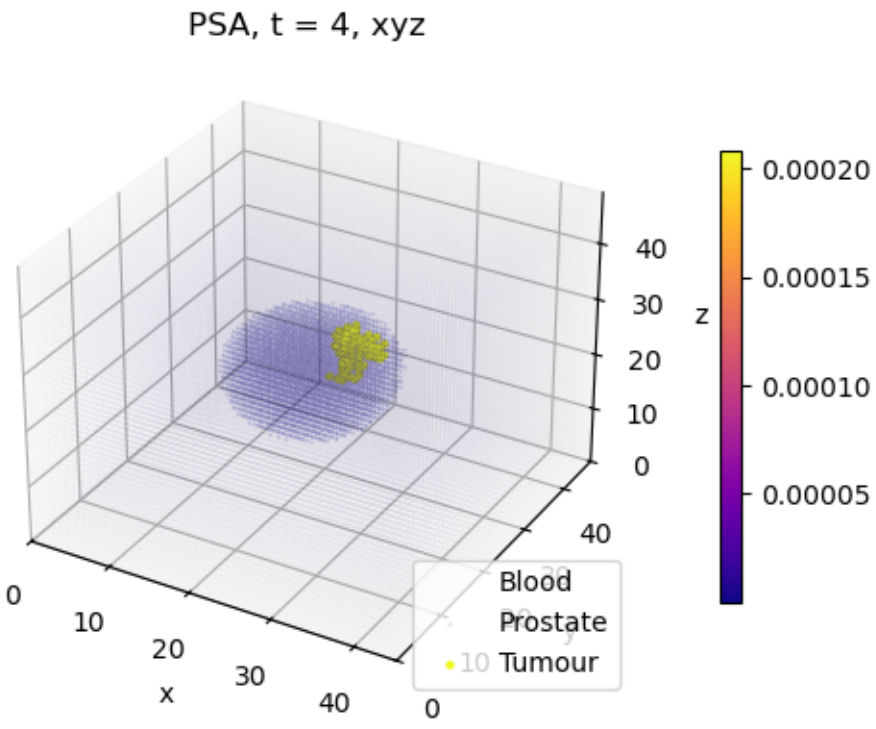

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2

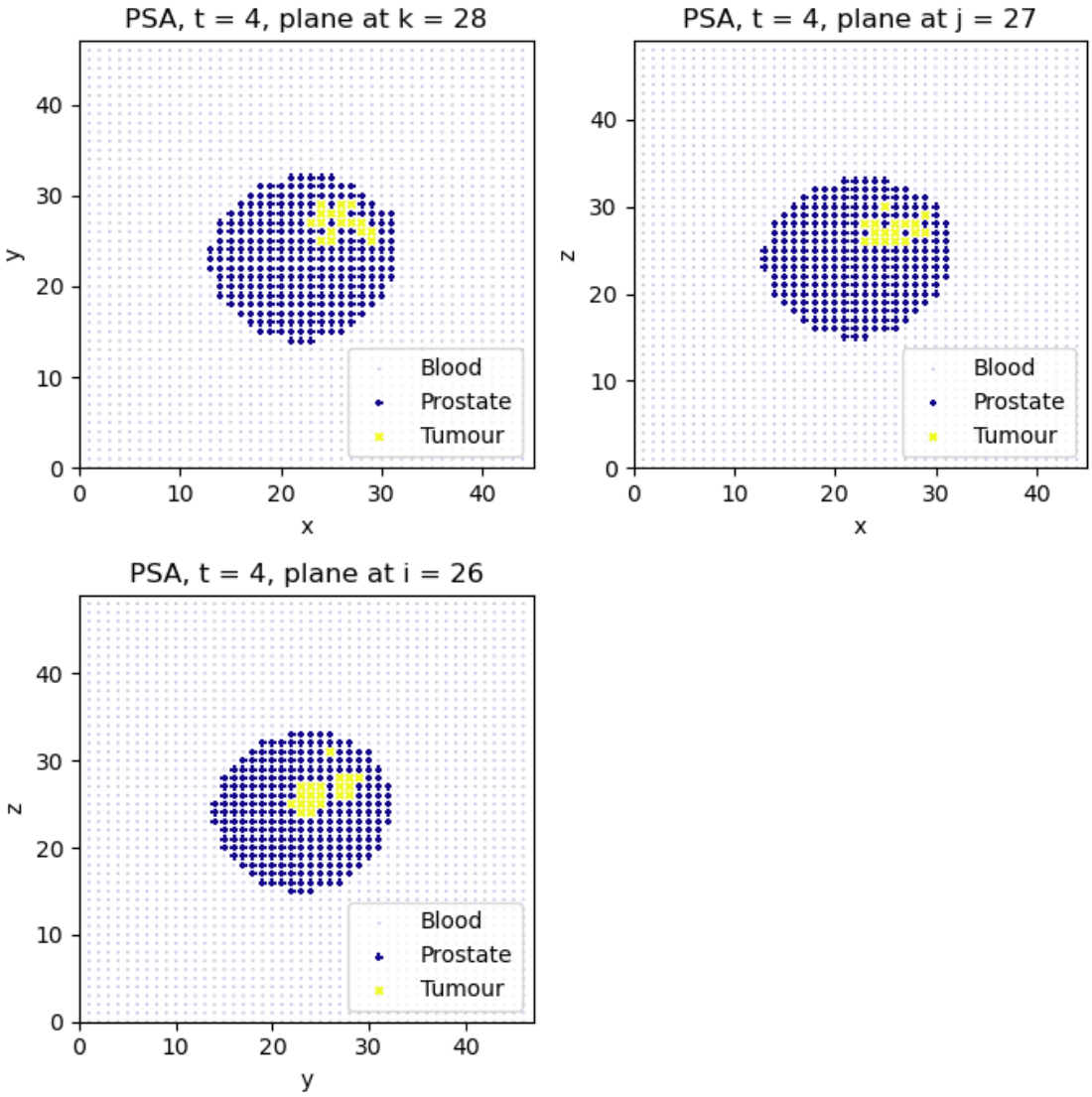

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2

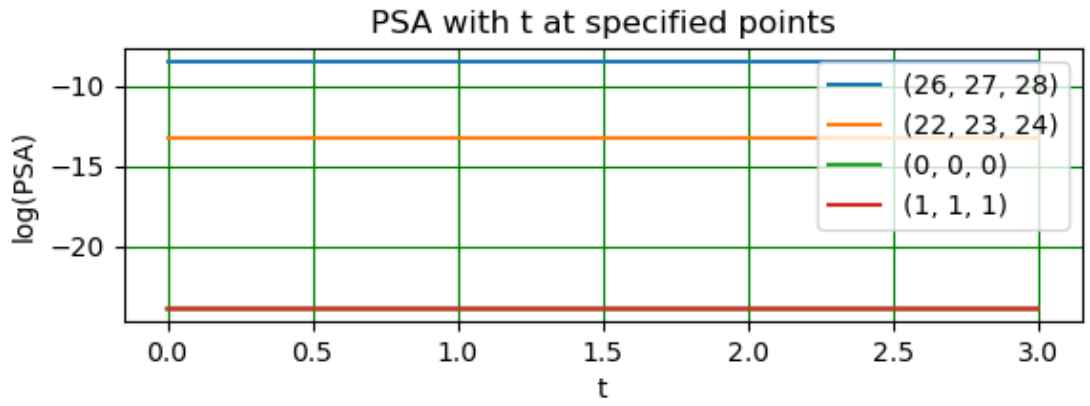

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2

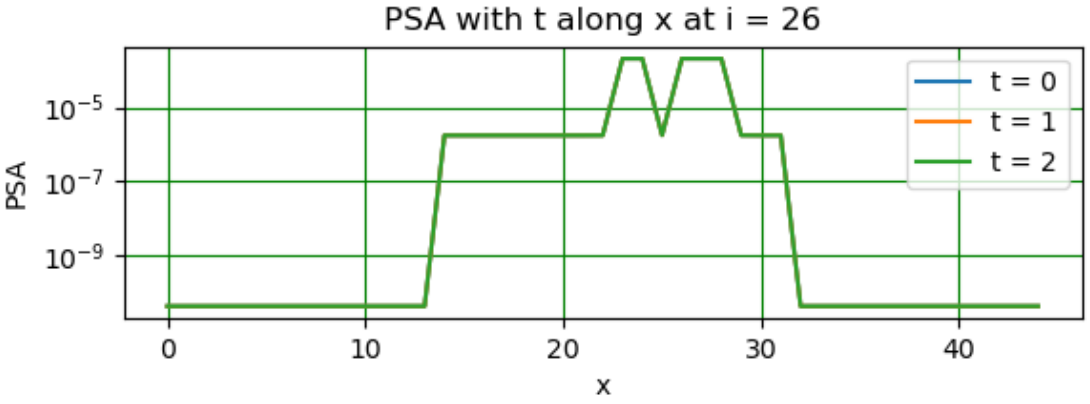


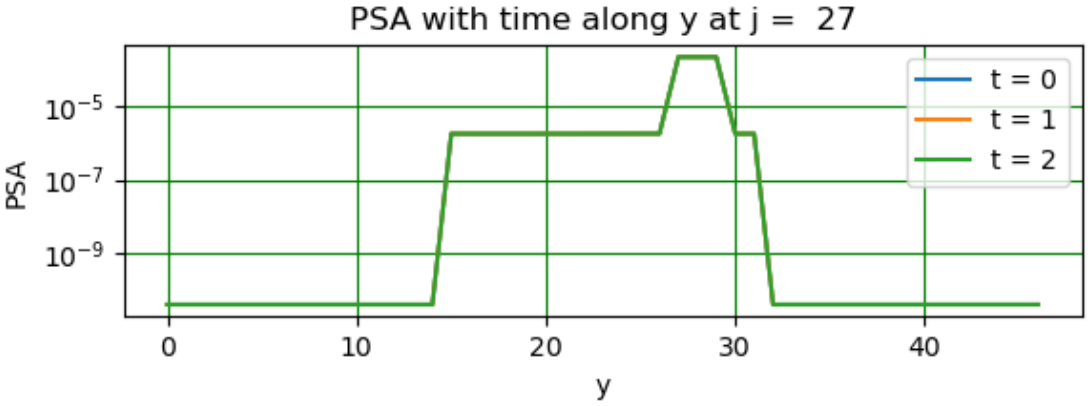

```
At time 15:
No. tumour cells 66, prostate cells 4138, blood cells 99431
PSA_top_level 4.135951e-11
-------------------------------------------------------------------------------
--------------------------
Time 16
Diffusion PSA_top, with tumour
Diffusion converged, nt = 4, ext_test = 1.534796e-05, prostate_test =
5.845445e-04
```

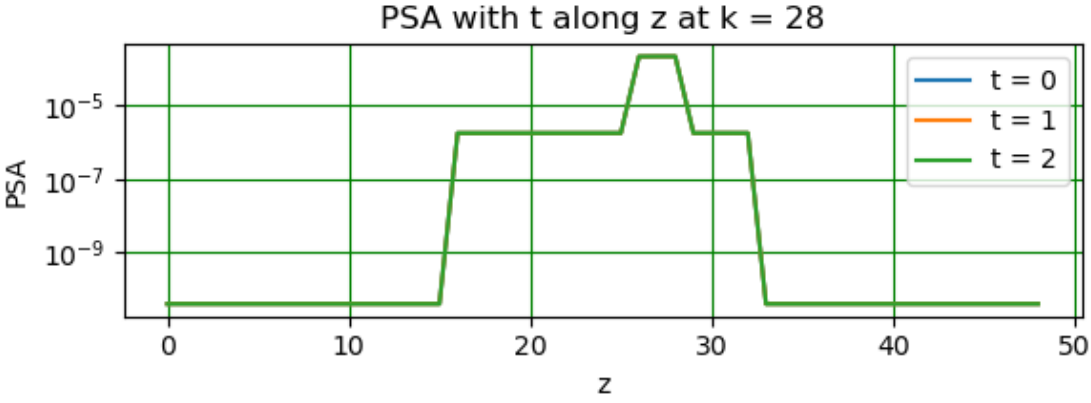

PSA, t = 4, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

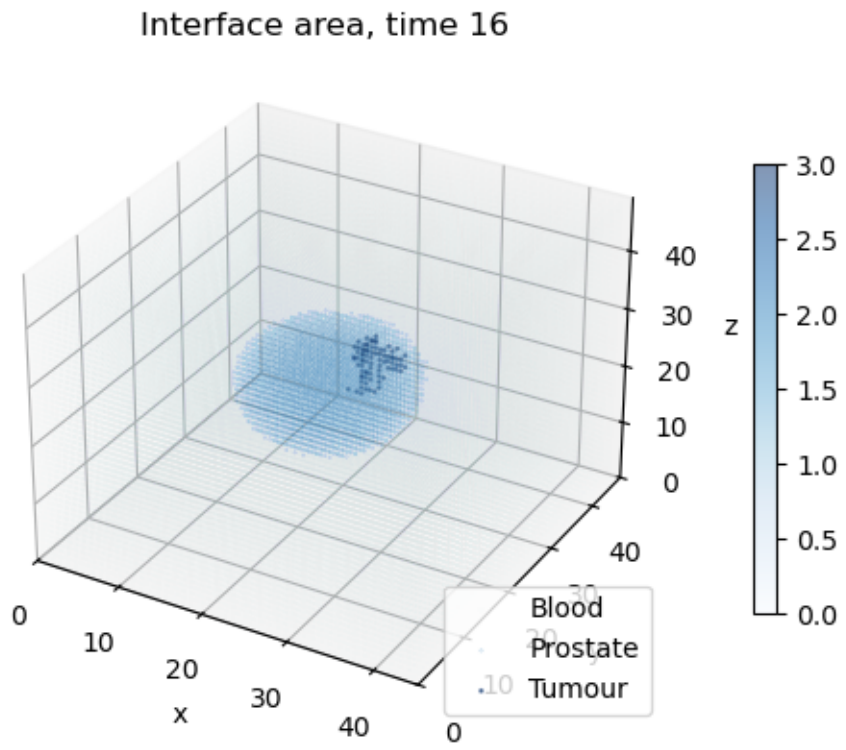

PSA, t = 4, plane at k = 28
PSA, t = 4, plane at j = 27
PSA, t = 4, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−10
−15
−20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)

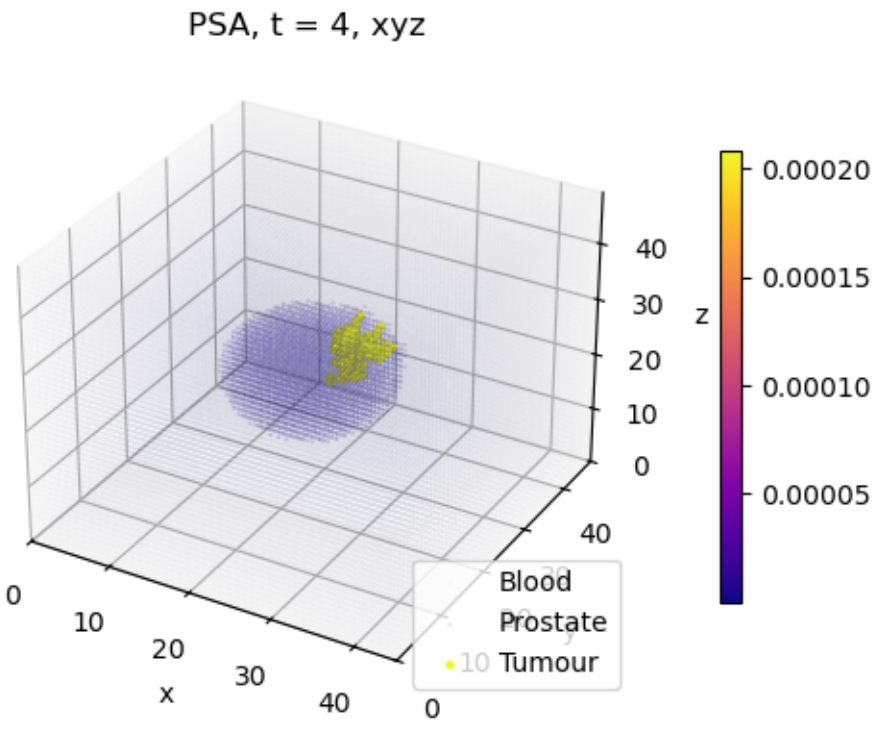

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2

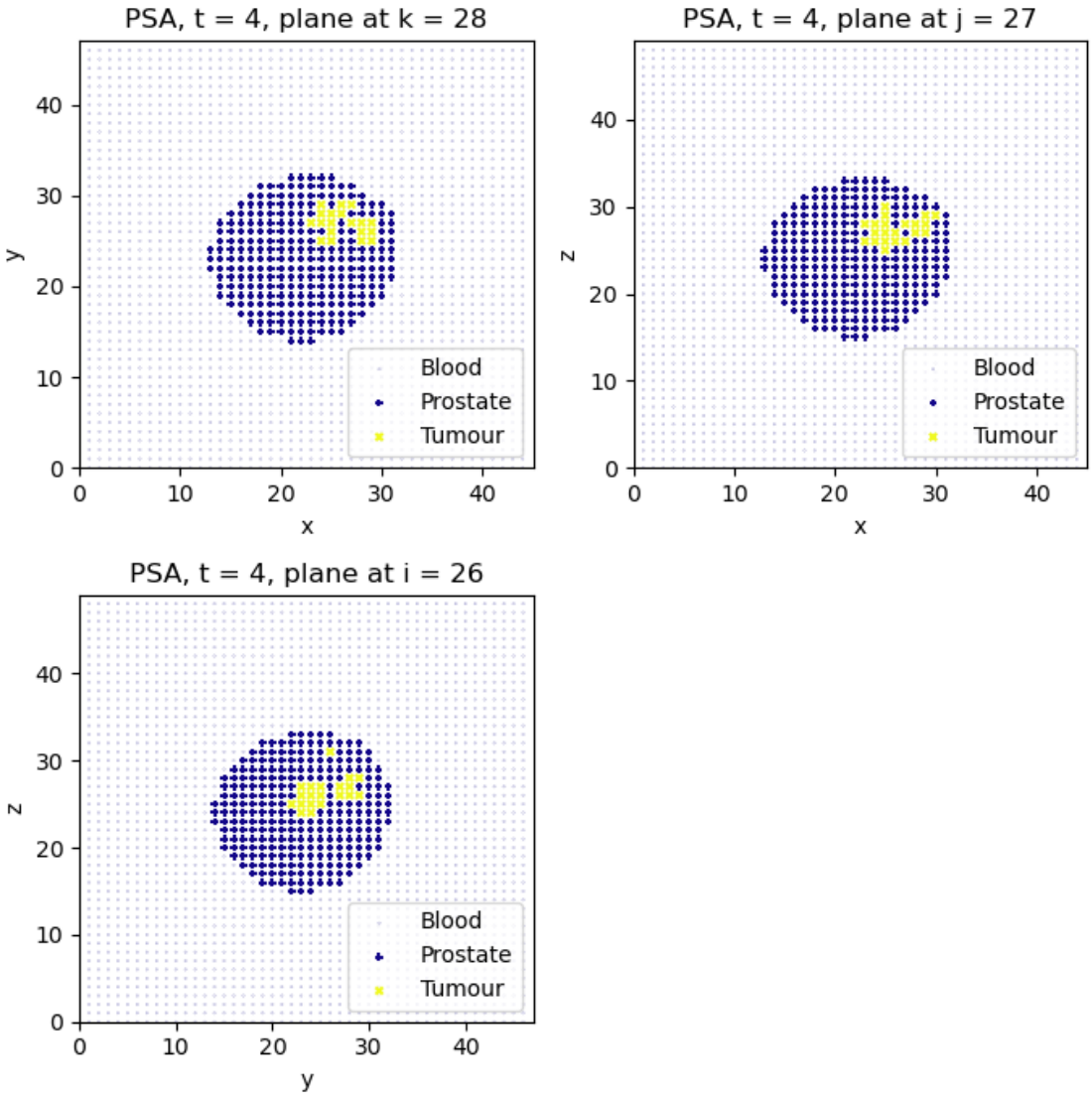

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2

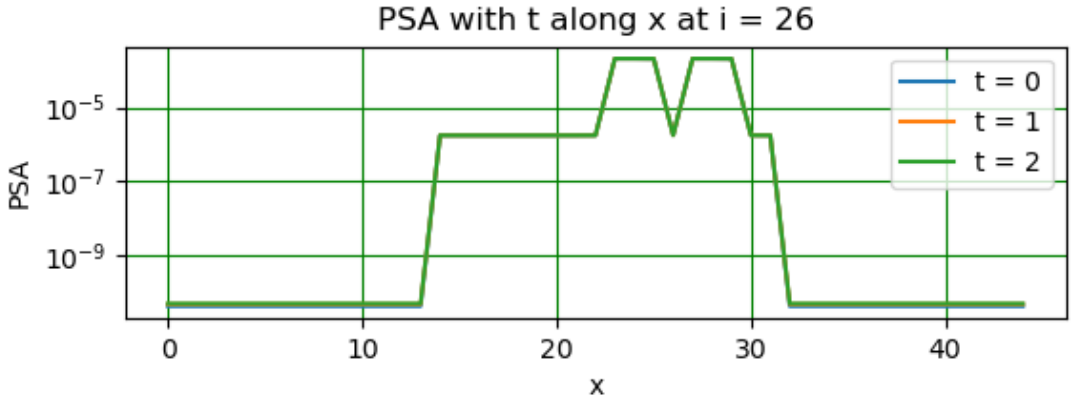


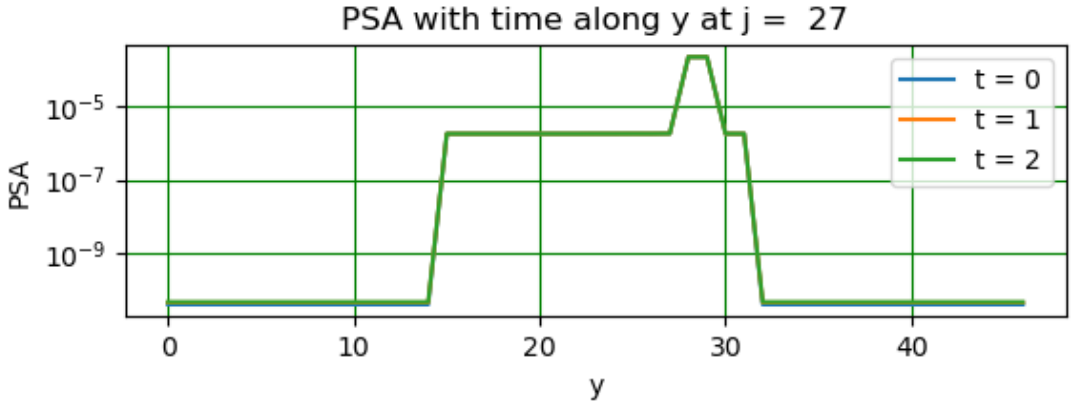

```
At time 16:
No. tumour cells 82, prostate cells 4138, blood cells 99415
PSA_top_level 4.126657e-11
-------------------------------------------------------------------------------
---------------------------
Time 17
Diffusion PSA_top, with tumour
Diffusion converged, nt = 4, ext_test = 9.685887e-04, prostate_test =
4.726574e-04
```

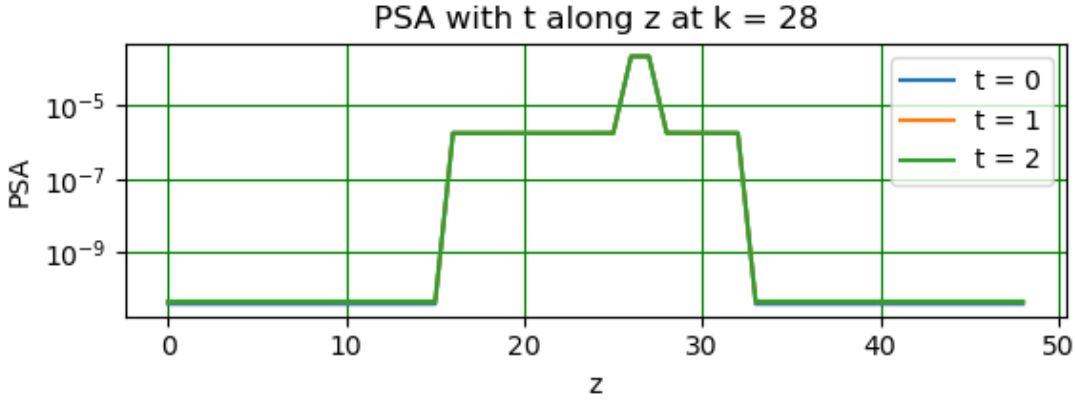

PSA, t = 4, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

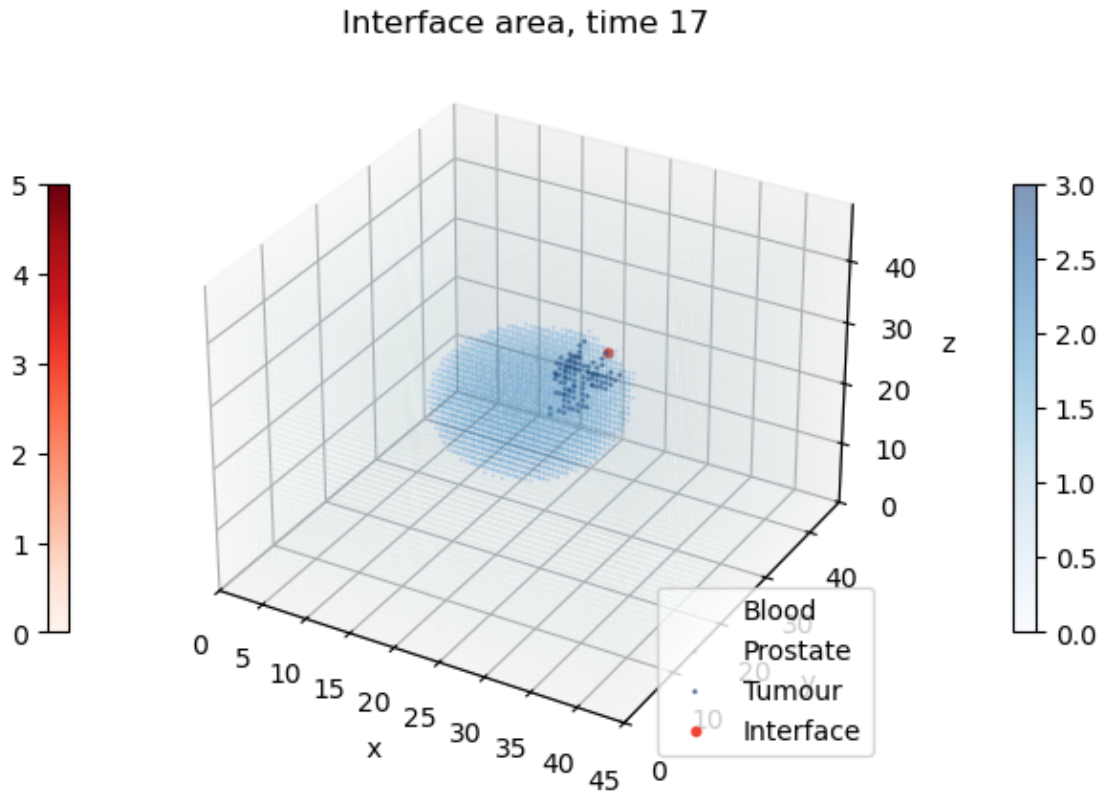

PSA, t = 4, plane at k = 28
PSA, t = 4, plane at j = 27
PSA, t = 4, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)

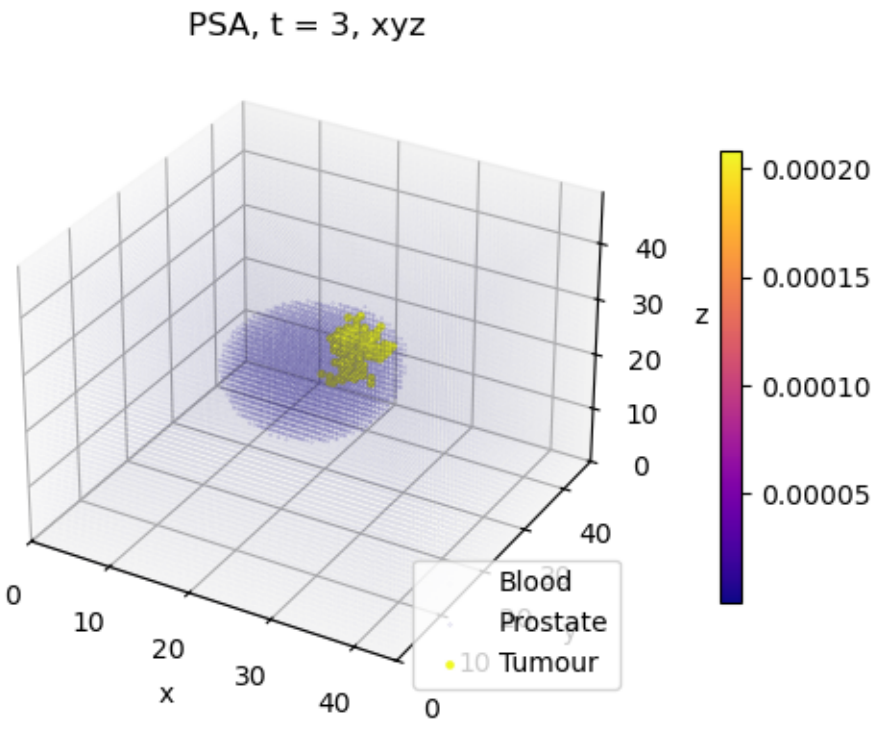

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2

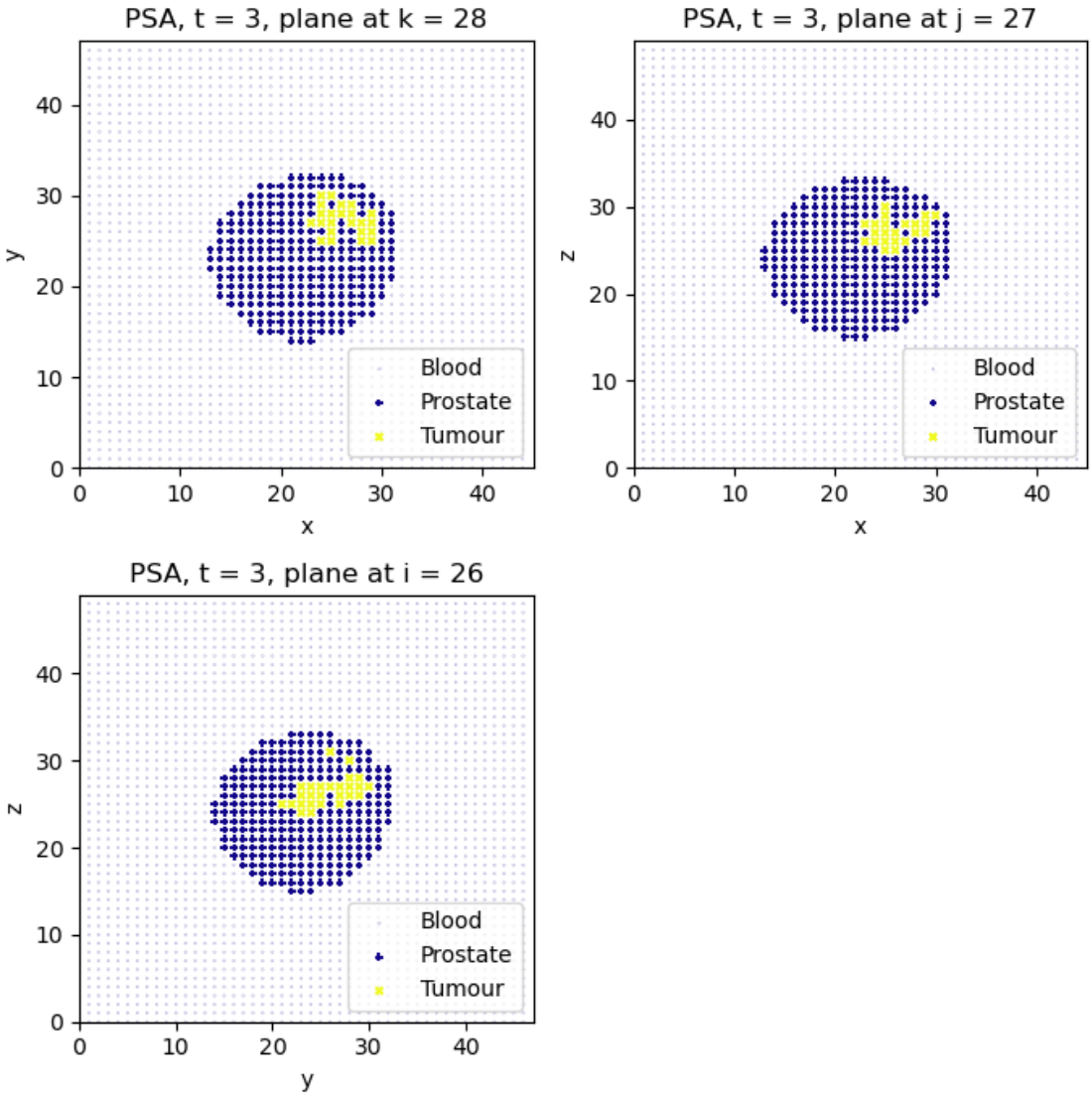

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2

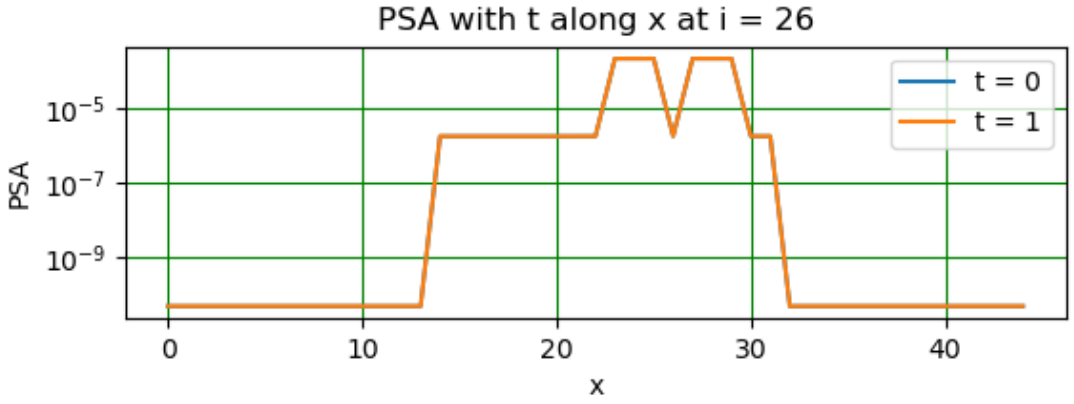


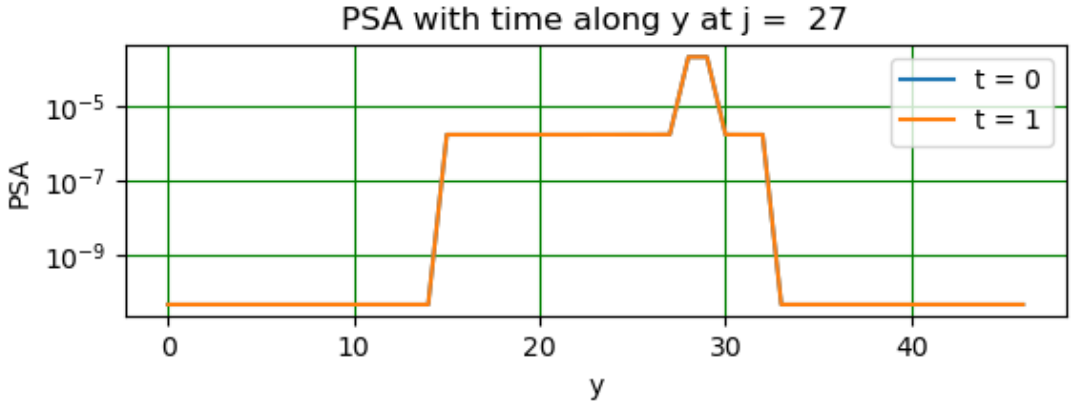

```
At time 17:
No. tumour cells 102, prostate cells 4138, blood cells 99395
PSA_top_level 4.913413e-11
--------------------------------------------------------------------------------
----------------------------
Time 18
Diffusion PSA_top, with tumour
Diffusion converged, nt = 3, ext_test = 7.490770e-05, prostate_test =
7.072289e-04
```


PSA, t = 3, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

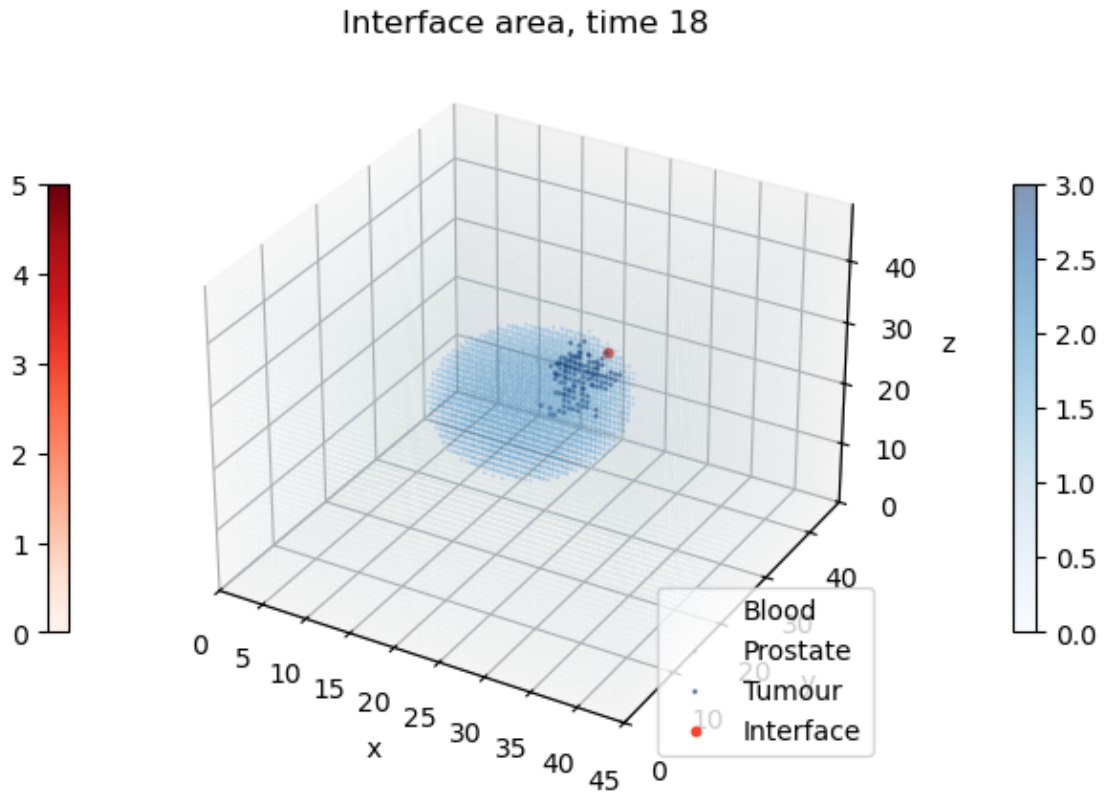

PSA, t = 3, plane at k = 28
PSA, t = 3, plane at j = 27
PSA, t = 3, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)

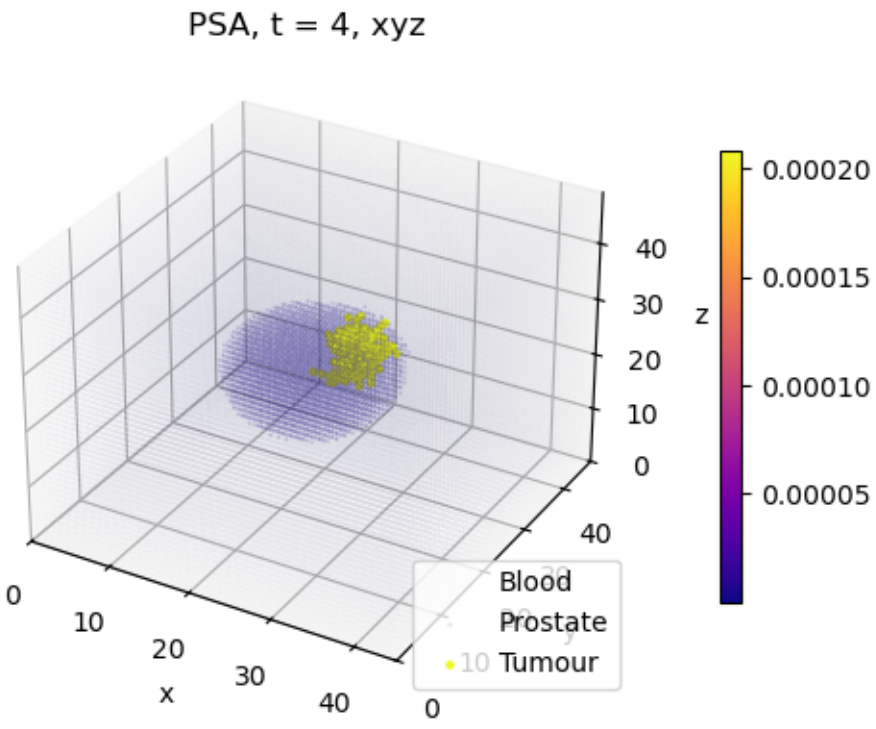

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1

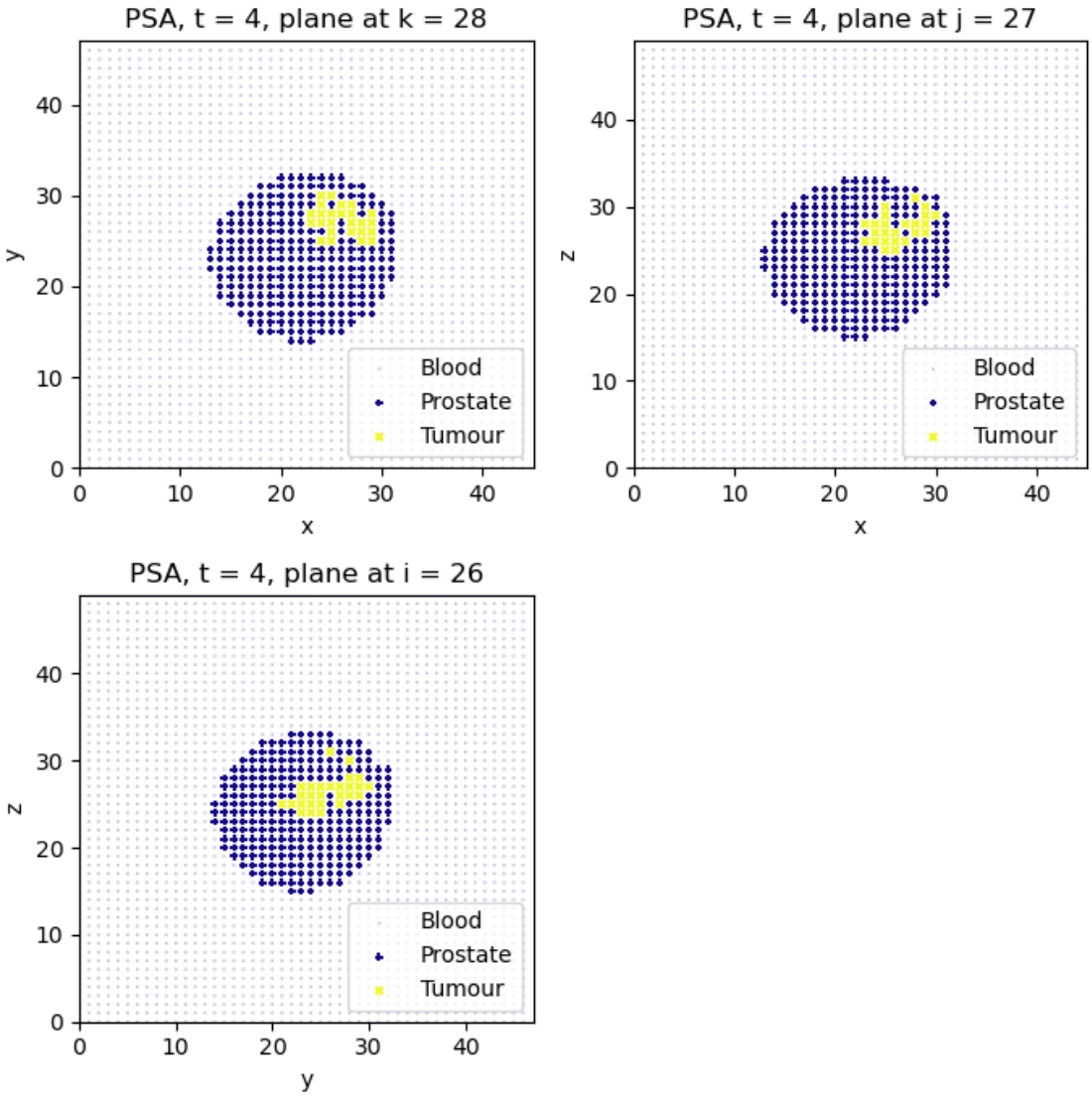

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1

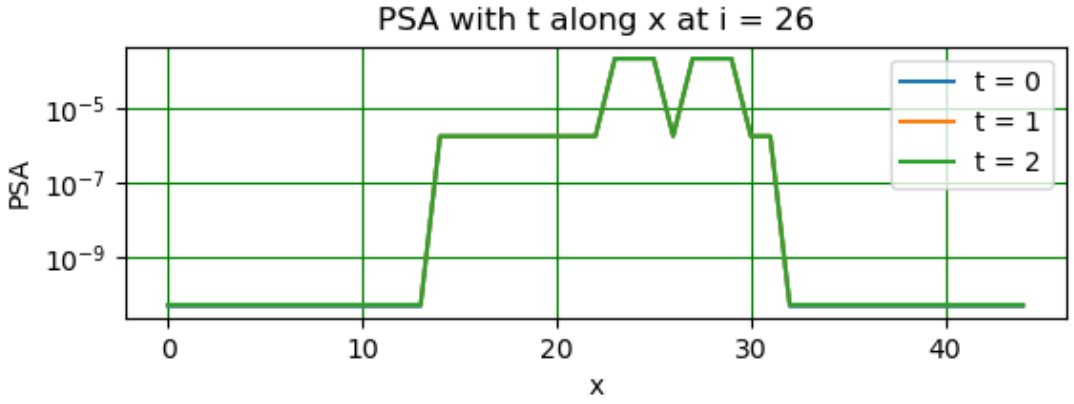


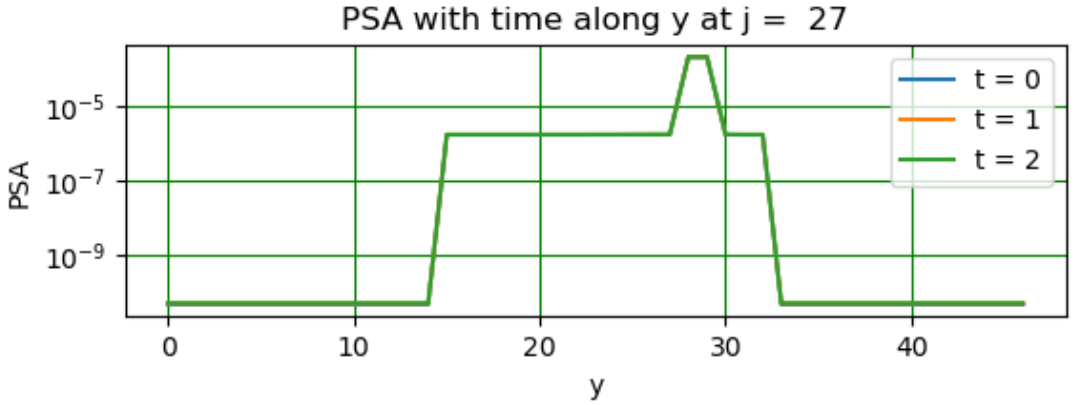

```
At time 18:
No. tumour cells 126, prostate cells 4138, blood cells 99371
PSA_top_level 4.933062e-11
--------------------------------------------------------------------------------
----------------------------
Time 19
Diffusion PSA_top, with tumour
Diffusion converged, nt = 4, ext_test = 7.121195e-04, prostate_test =
3.110708e-04
```

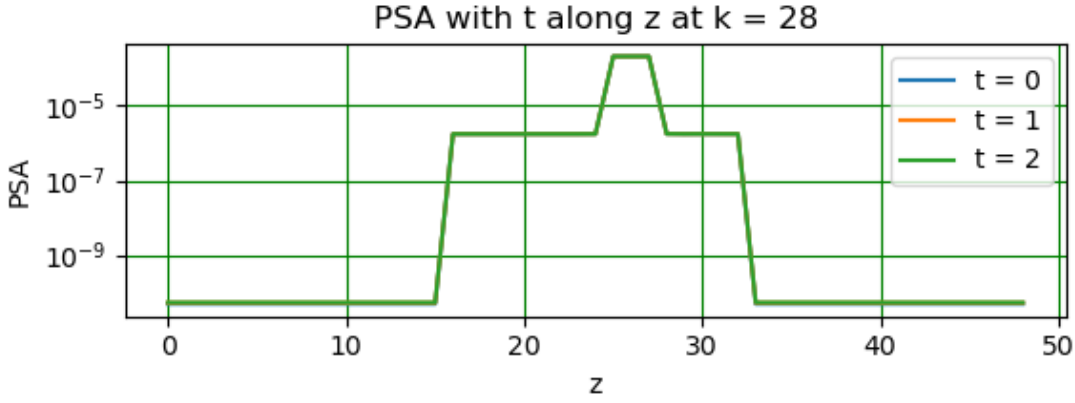

PSA, t = 4, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

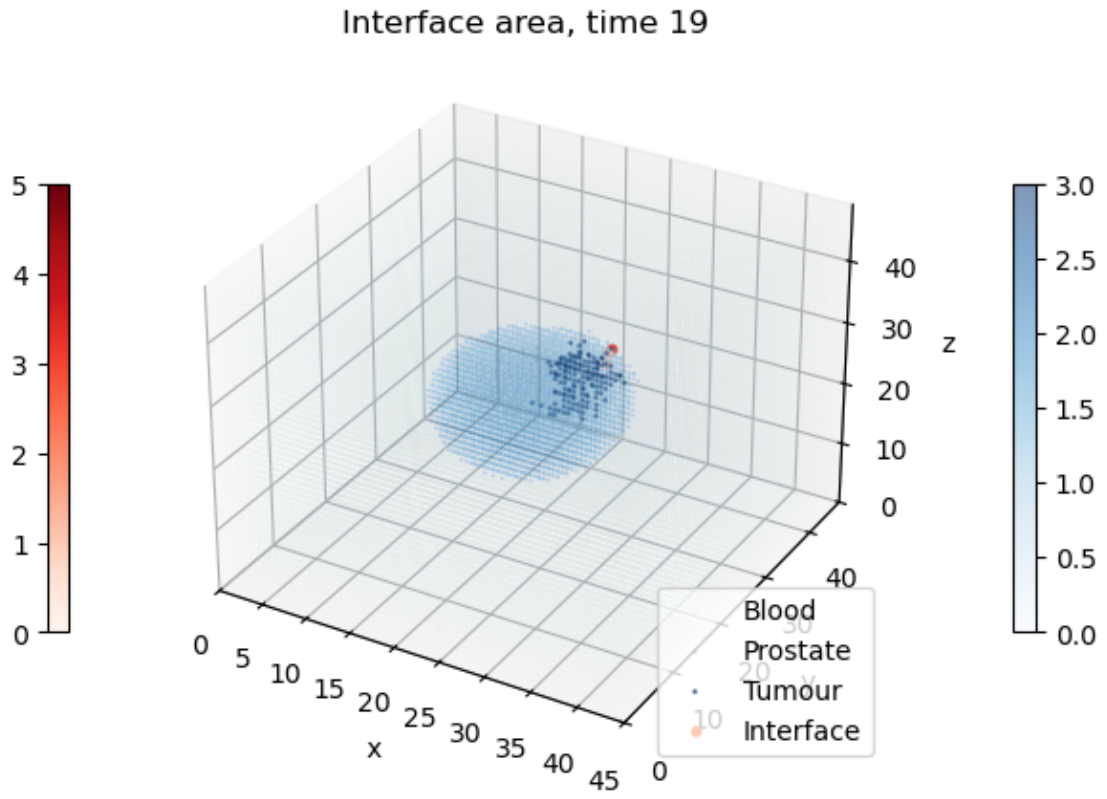

PSA, t = 4, plane at k = 28
PSA, t = 4, plane at j = 27
PSA, t = 4, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)

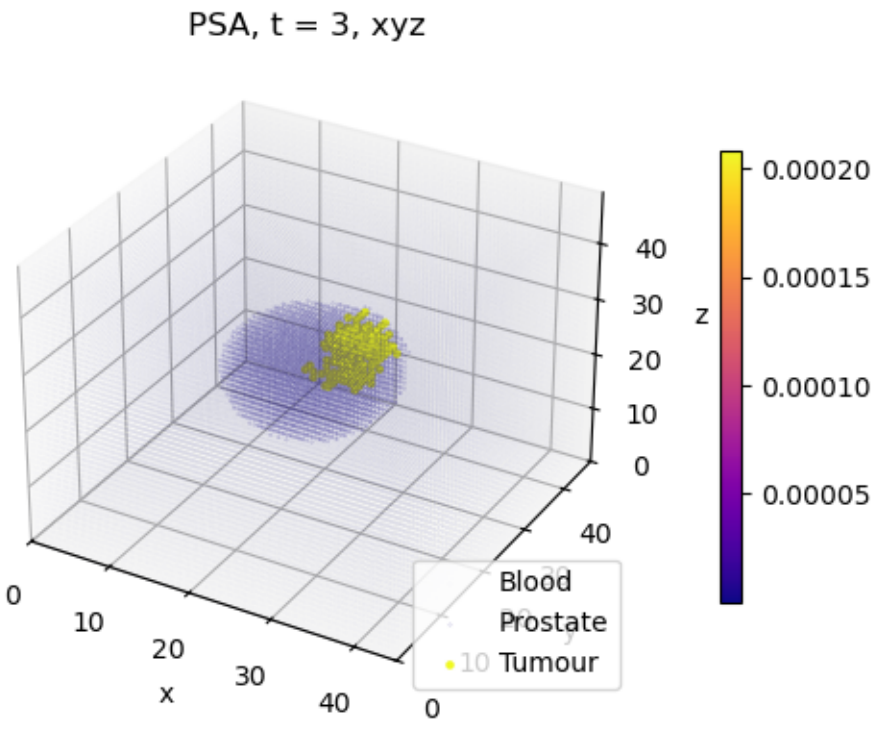

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2

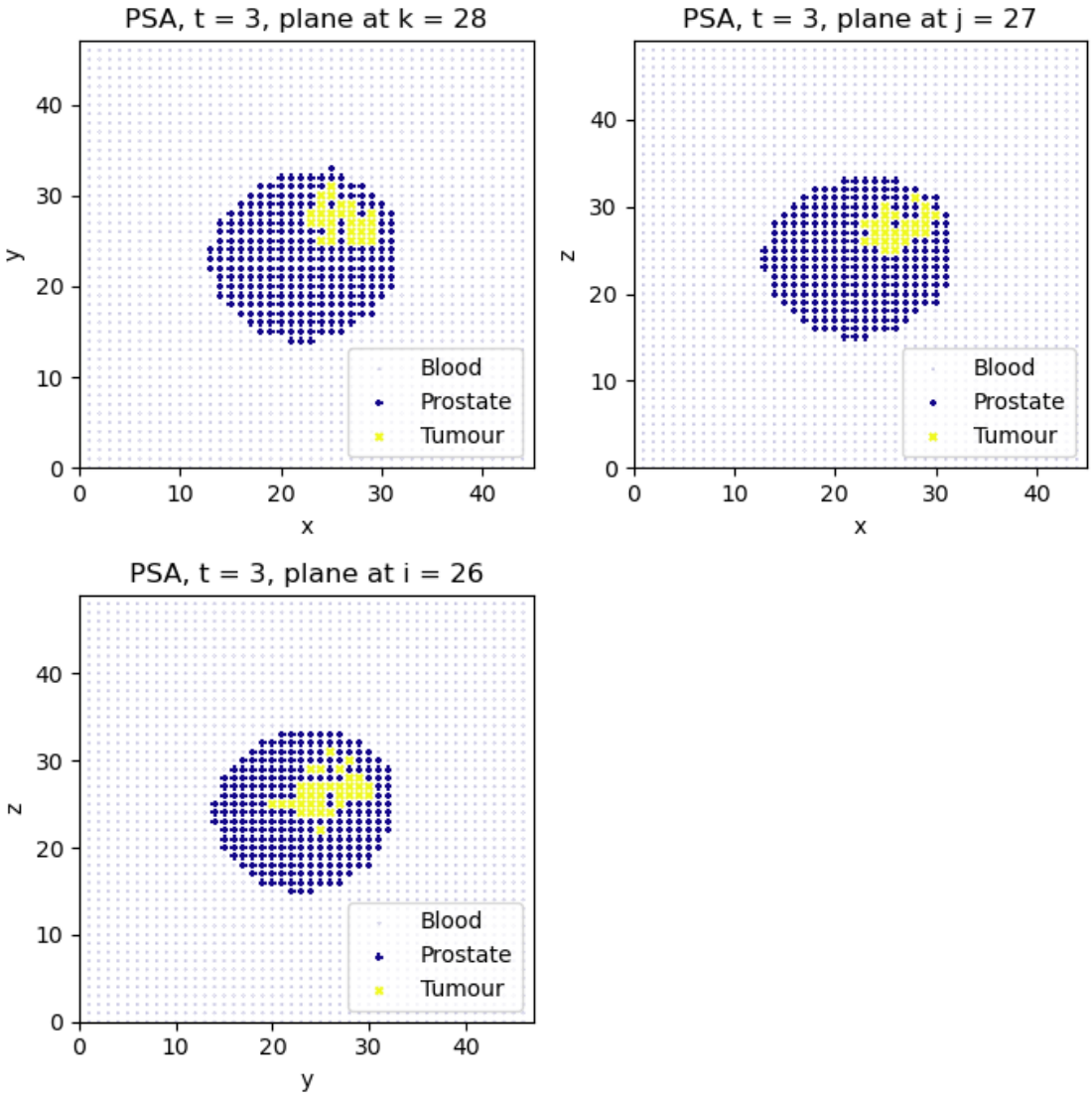

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2

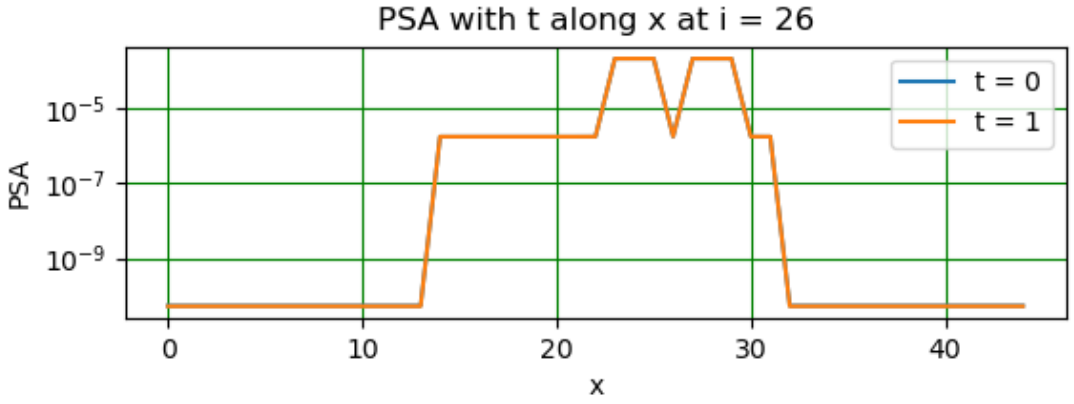


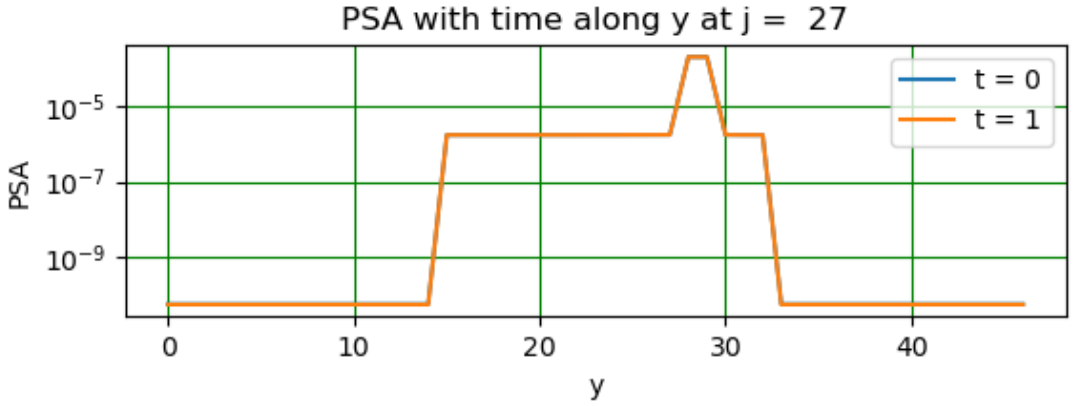

```
At time 19:
No. tumour cells 155, prostate cells 4138, blood cells 99342
PSA_top_level 5.439125e-11
------------------------------------------------------------------------------
---------------------------
Time 20
Diffusion PSA_top, with tumour
Diffusion converged, nt = 3, ext_test = 8.398081e-04, prostate_test =
4.677775e-04
```


PSA, t = 3, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

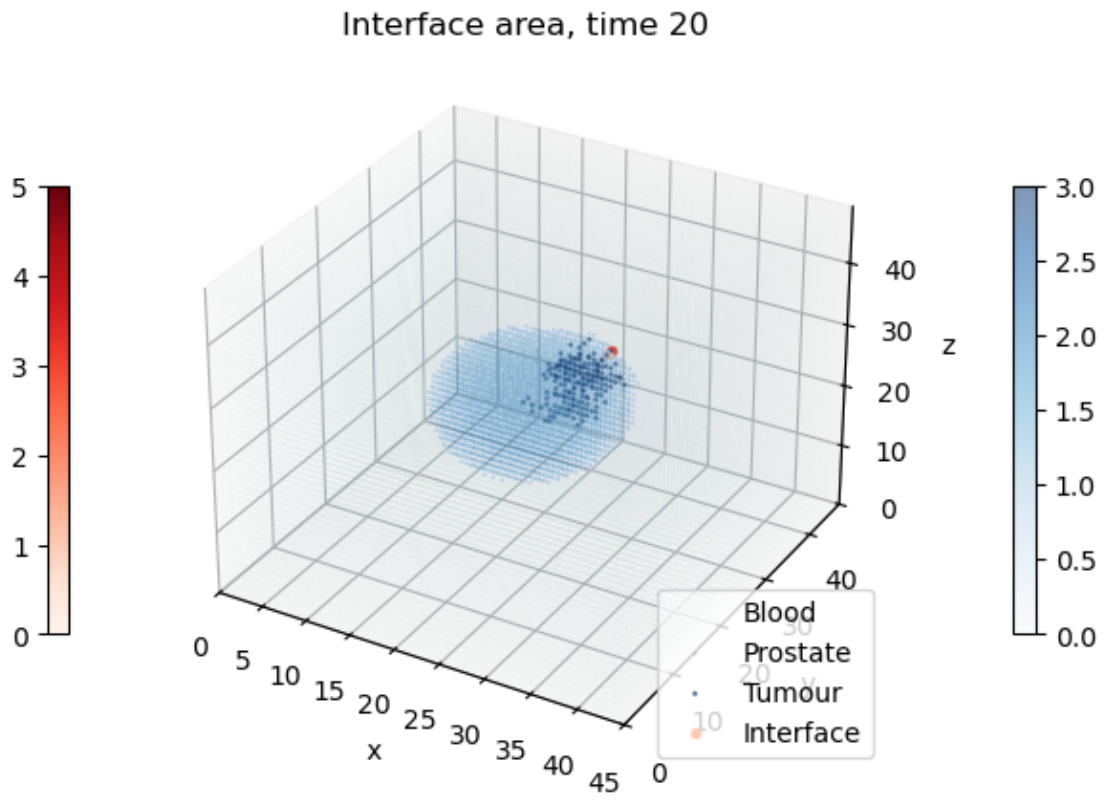

PSA, t = 3, plane at k = 28
PSA, t = 3, plane at j = 27
PSA, t = 3, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)

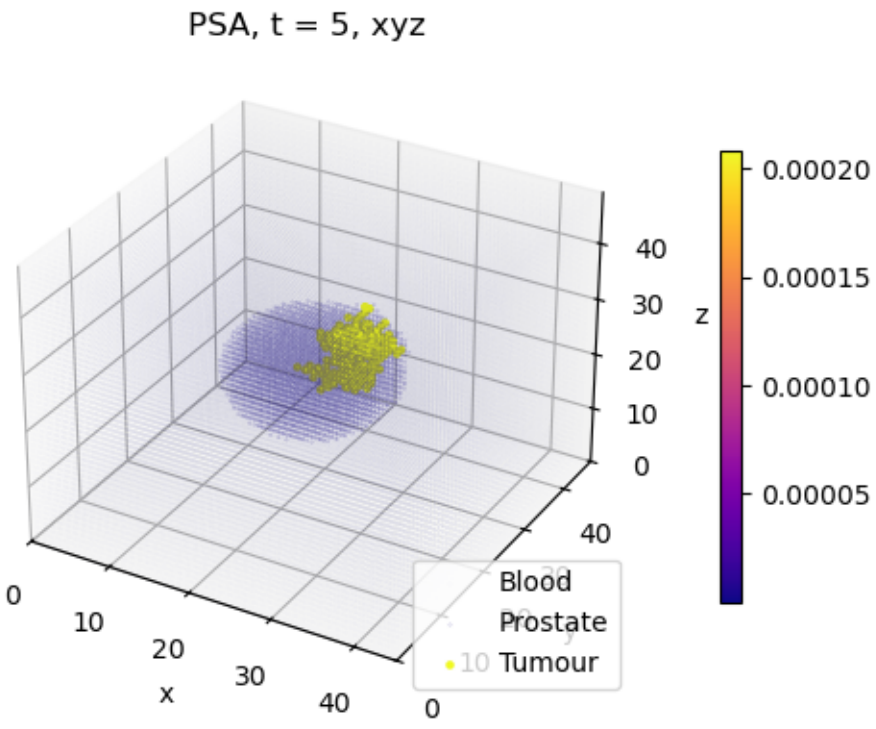

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1

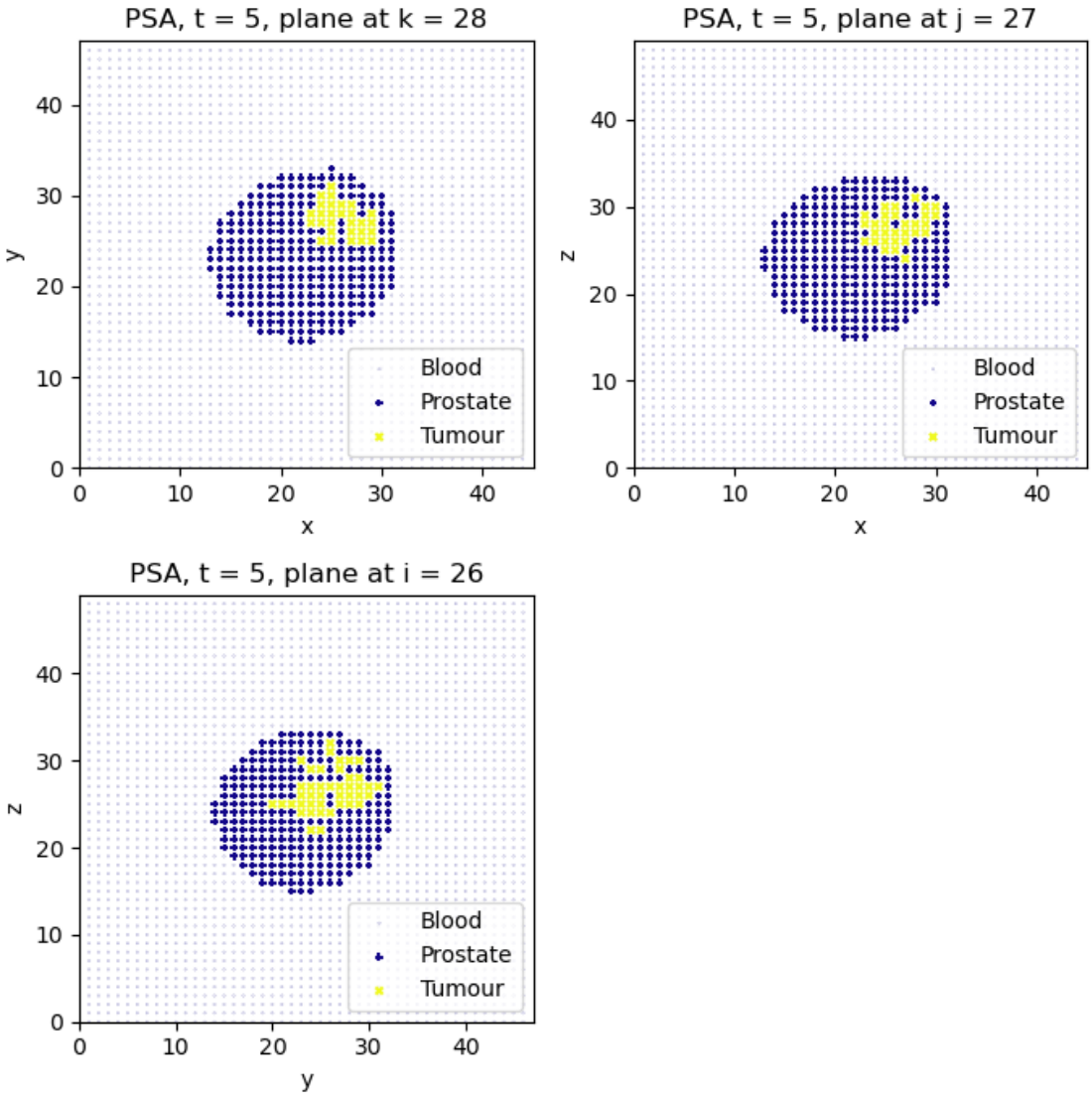

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1

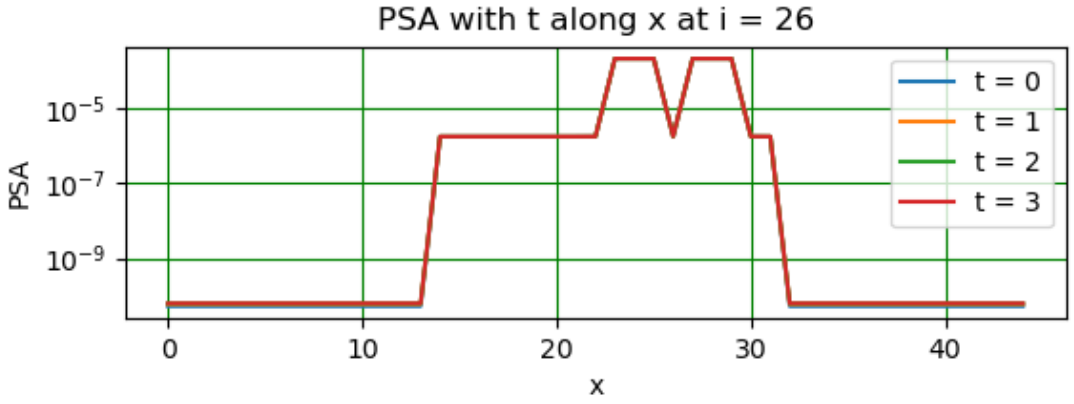


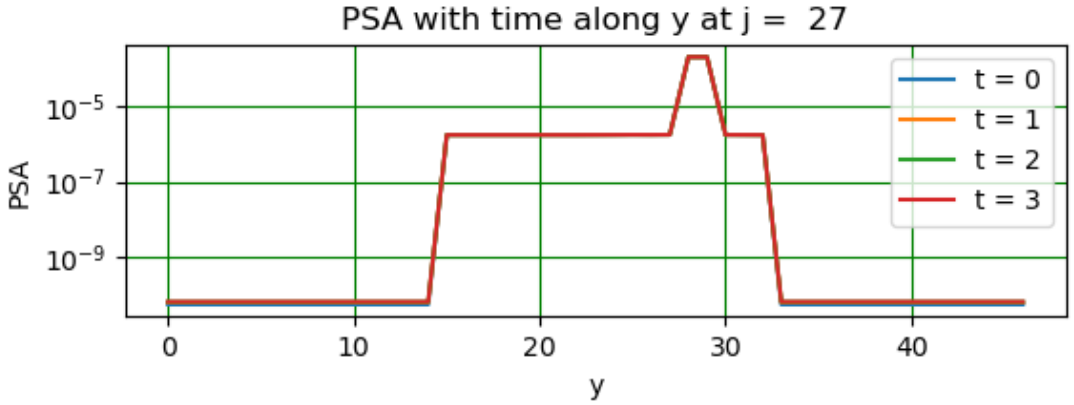

```
At time 20:
No. tumour cells 191, prostate cells 4138, blood cells 99306
PSA_top_level 5.219419e-11
--------------------------------------------------------------------------------
----------------------------
Time 21
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 3.590461e-04, prostate_test =
1.135930e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

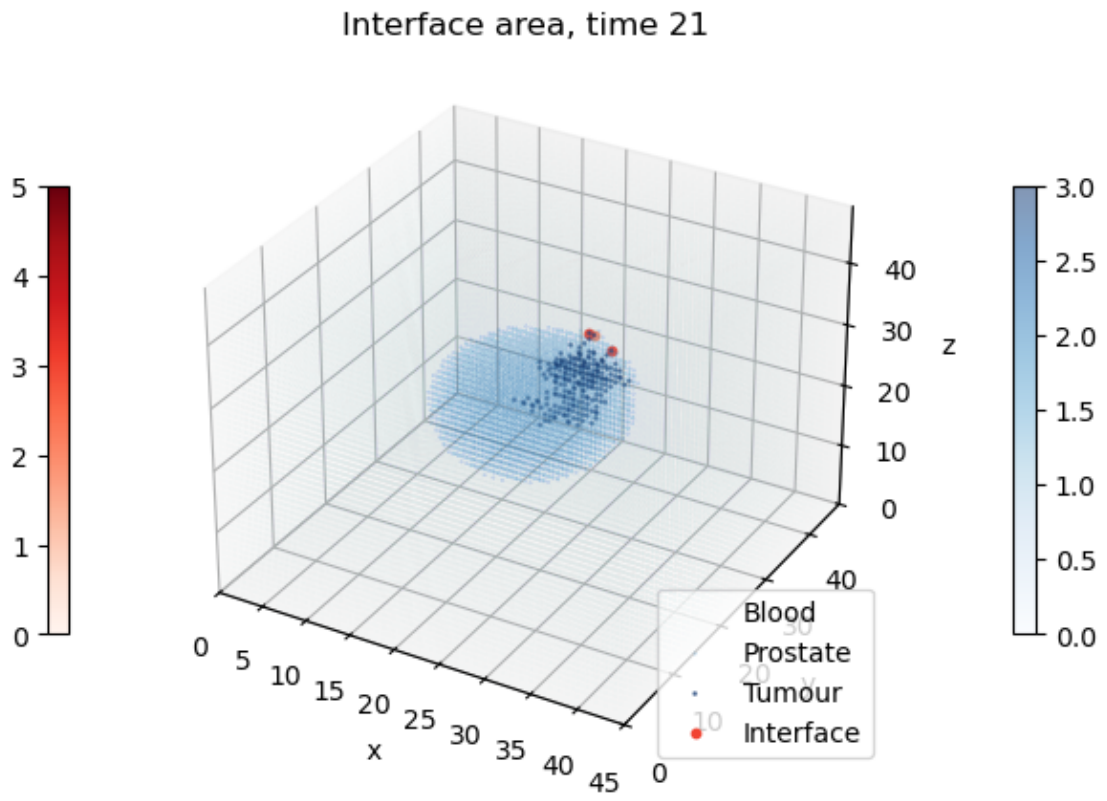

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

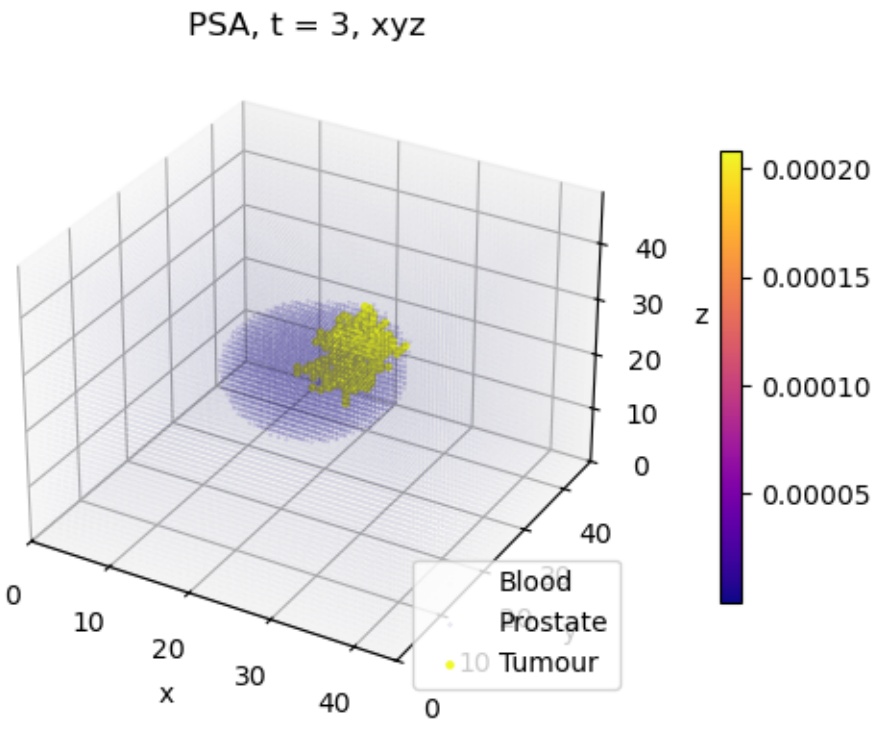

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

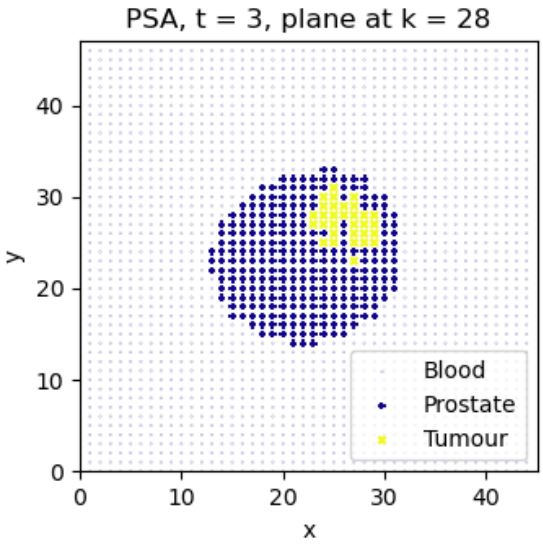

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3

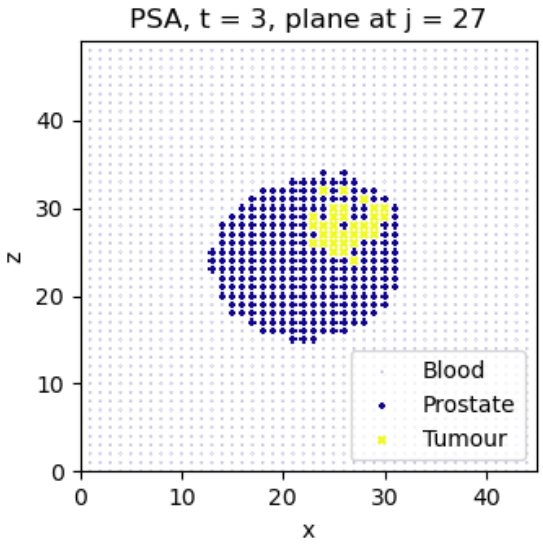

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

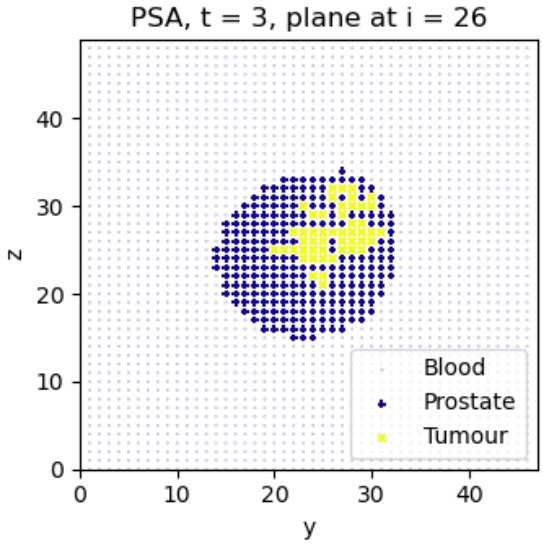


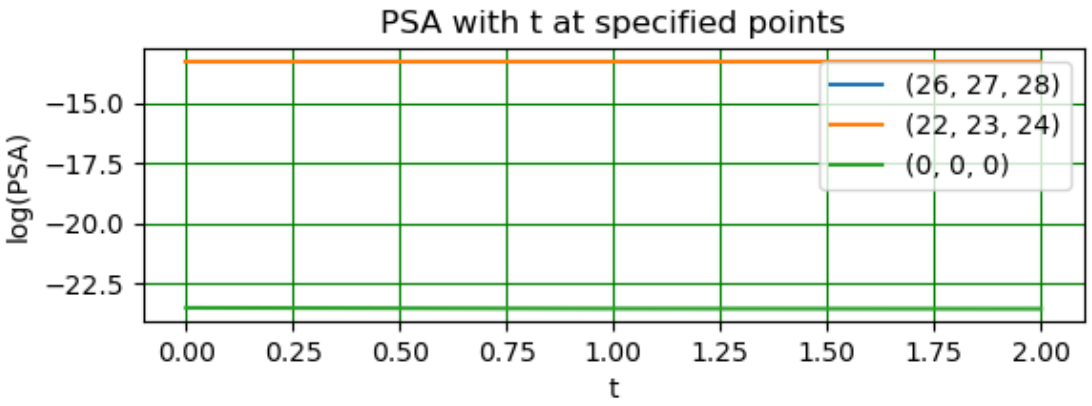

```
At time 21:
No. tumour cells 235, prostate cells 4138, blood cells 99262
PSA_top_level 6.228494e-11
--------------------------------------------------------------------------------
----------------------------
Time 22
Diffusion PSA_top, with tumour
Diffusion converged, nt = 3, ext_test = 8.815111e-04, prostate_test =
5.564585e-04
```

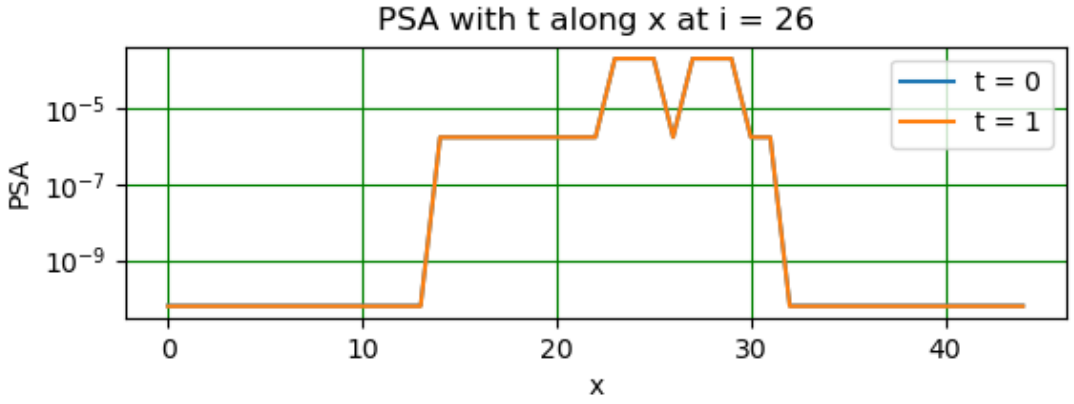

PSA, t = 3, xyz
0
10
20
30
40
x
0
40
30
20
10
0
z
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

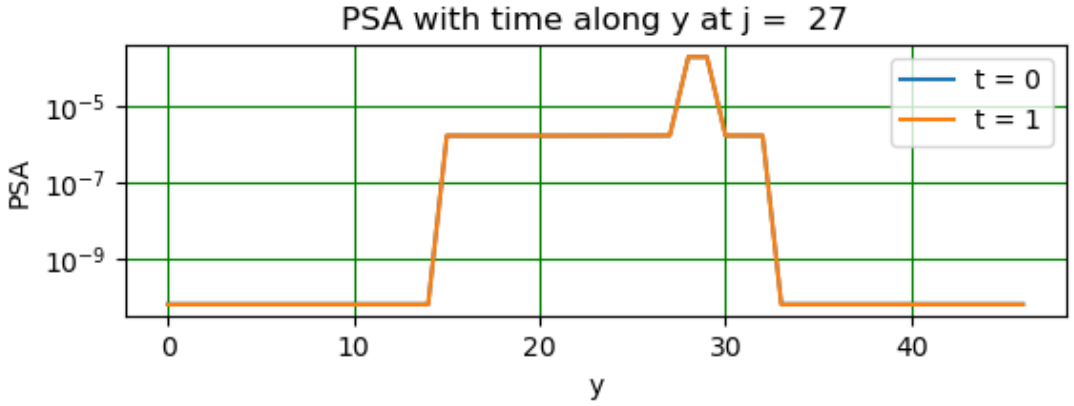

PSA, t = 3, plane at k = 28
y
x
Blood
Prostate
Tumour

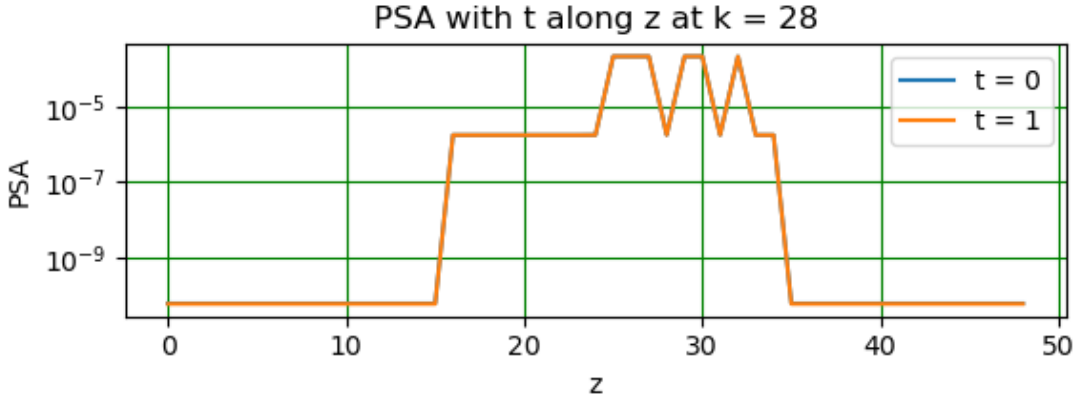

PSA, t = 3, plane at j = 27
z
x
Blood
Prostate
Tumour

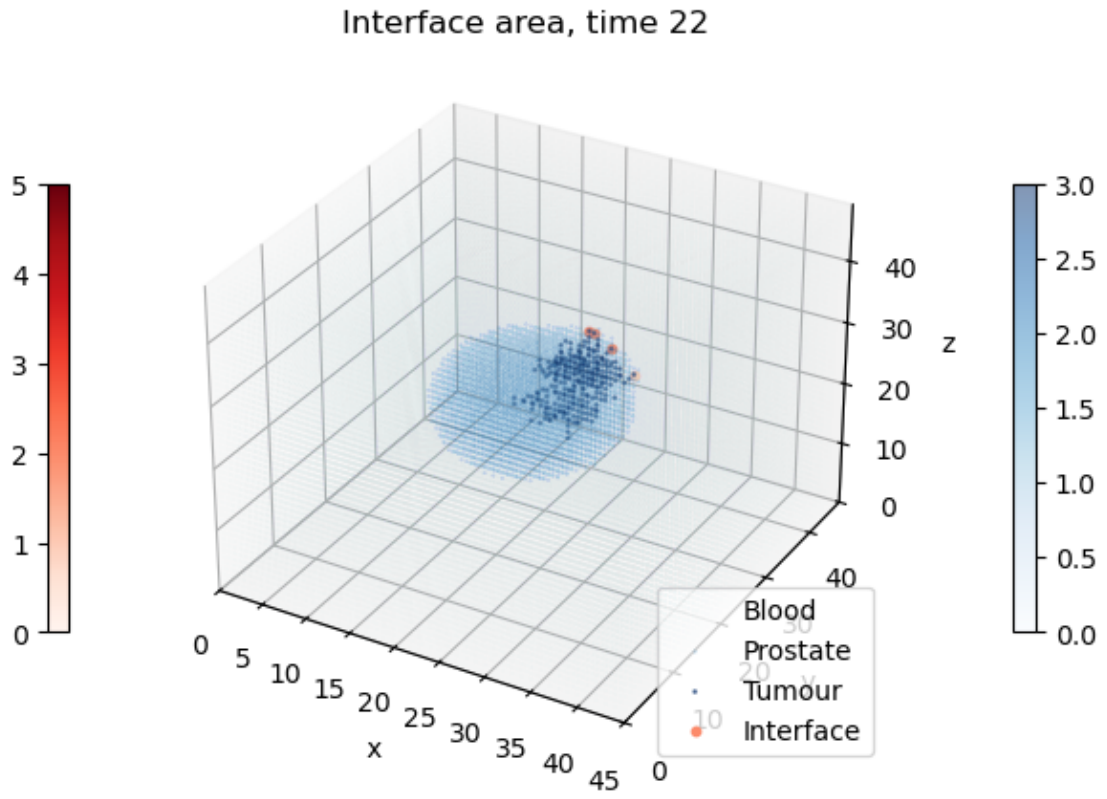

PSA, t = 3, plane at i = 26
z
y
Blood
Prostate
Tumour


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)

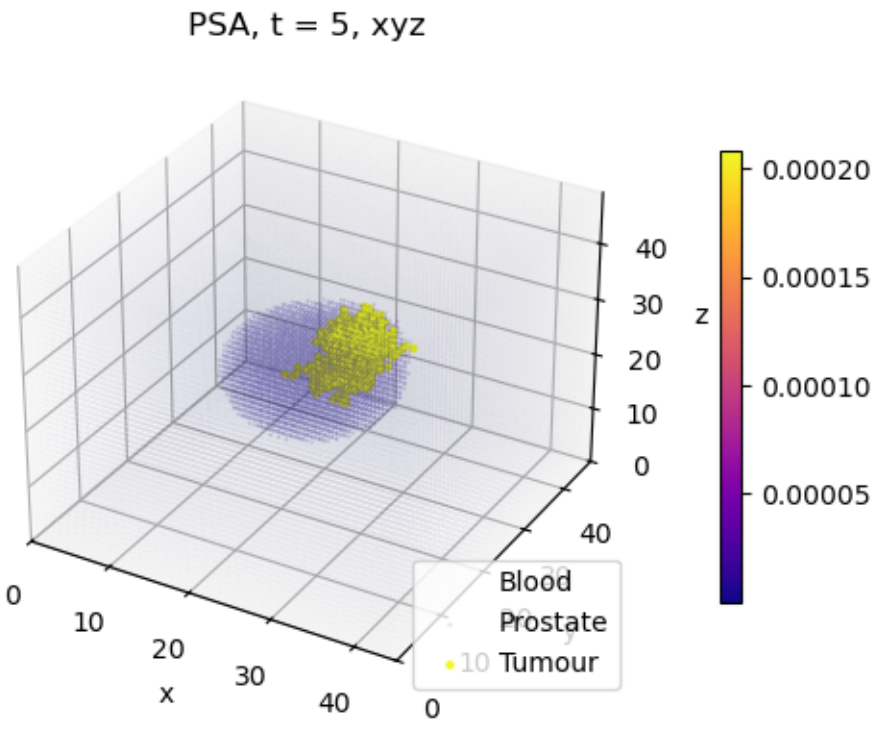

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1

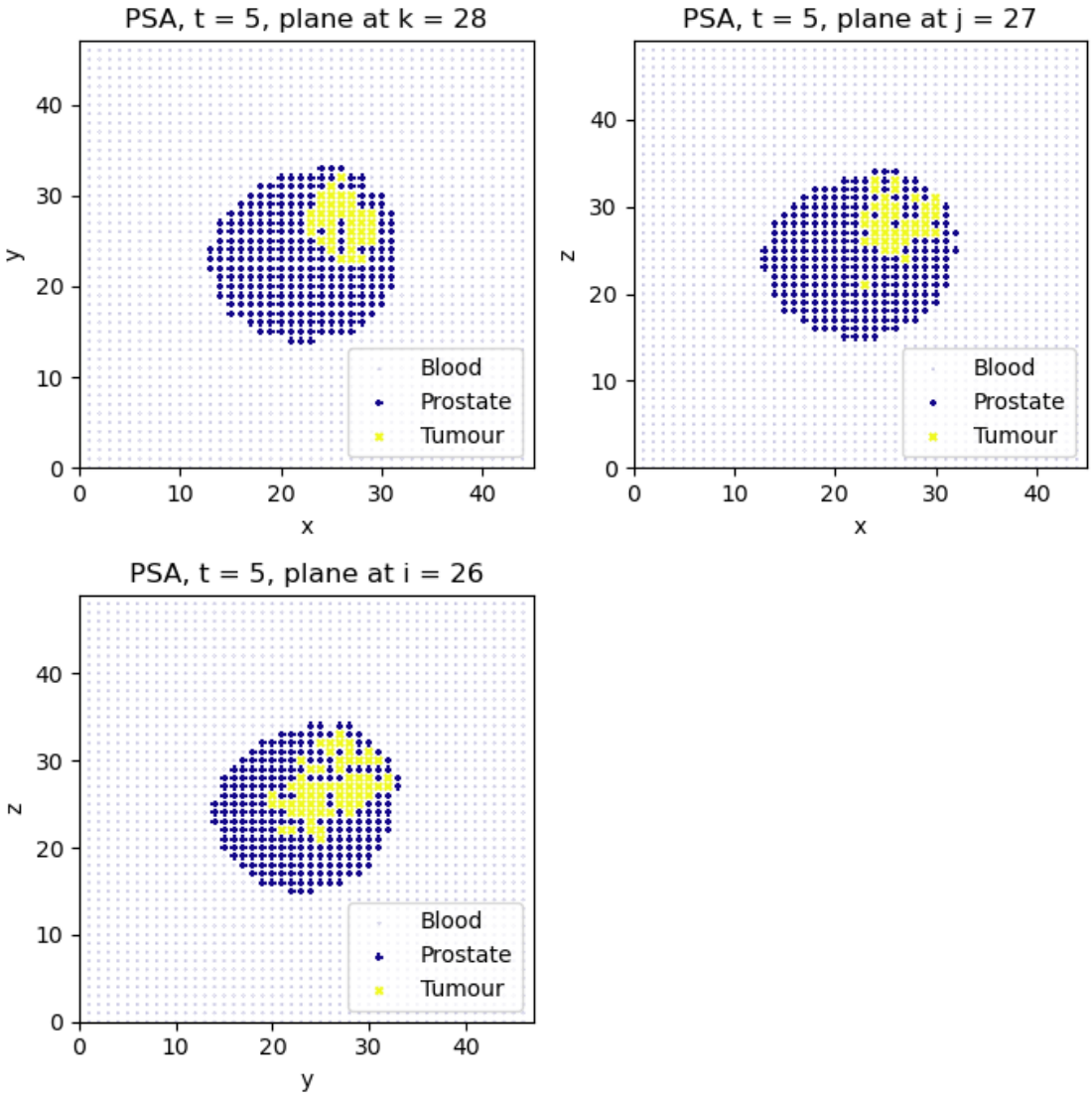

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1

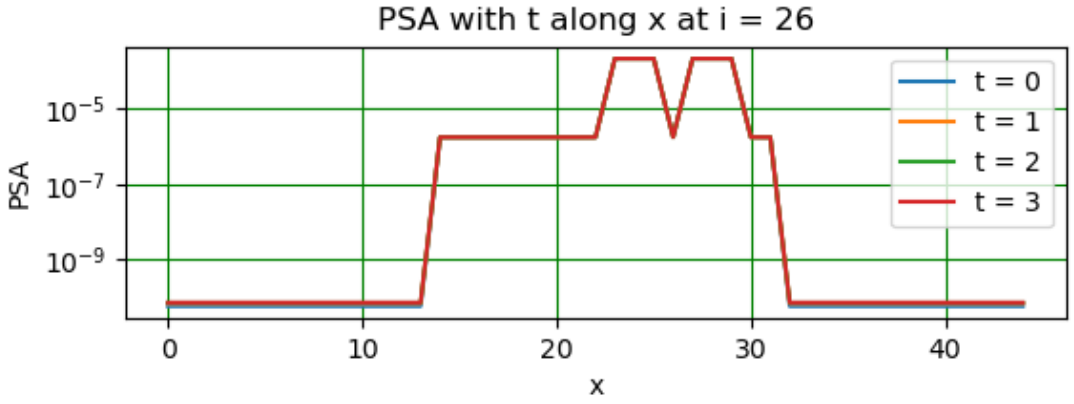


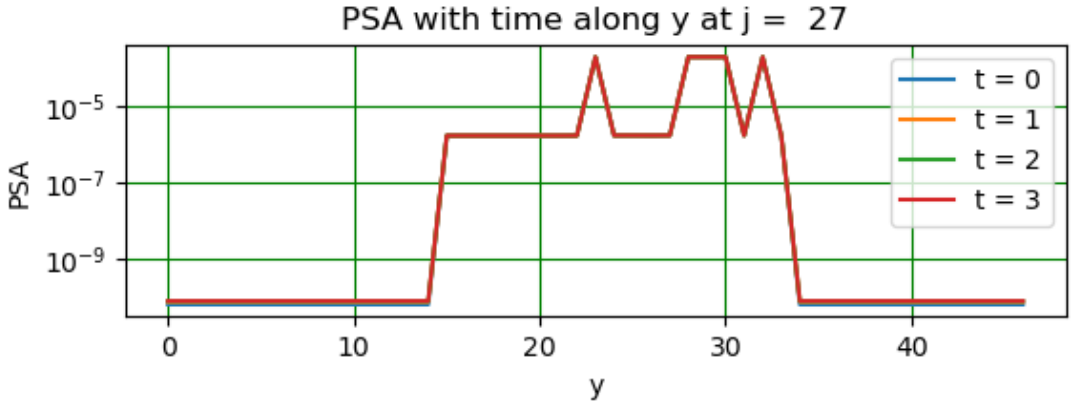

```
At time 22:
No. tumour cells 289, prostate cells 4138, blood cells 99208
PSA_top_level 5.957494e-11
--------------------------------------------------------------------------------
----------------------------
Time 23
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 4.301075e-04, prostate_test =
1.627675e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
10
20
30
40
z
0
10
20
30
40
y
Blood
Prostate
Tumour
0.00020
0.00015
0.00010
0.00005

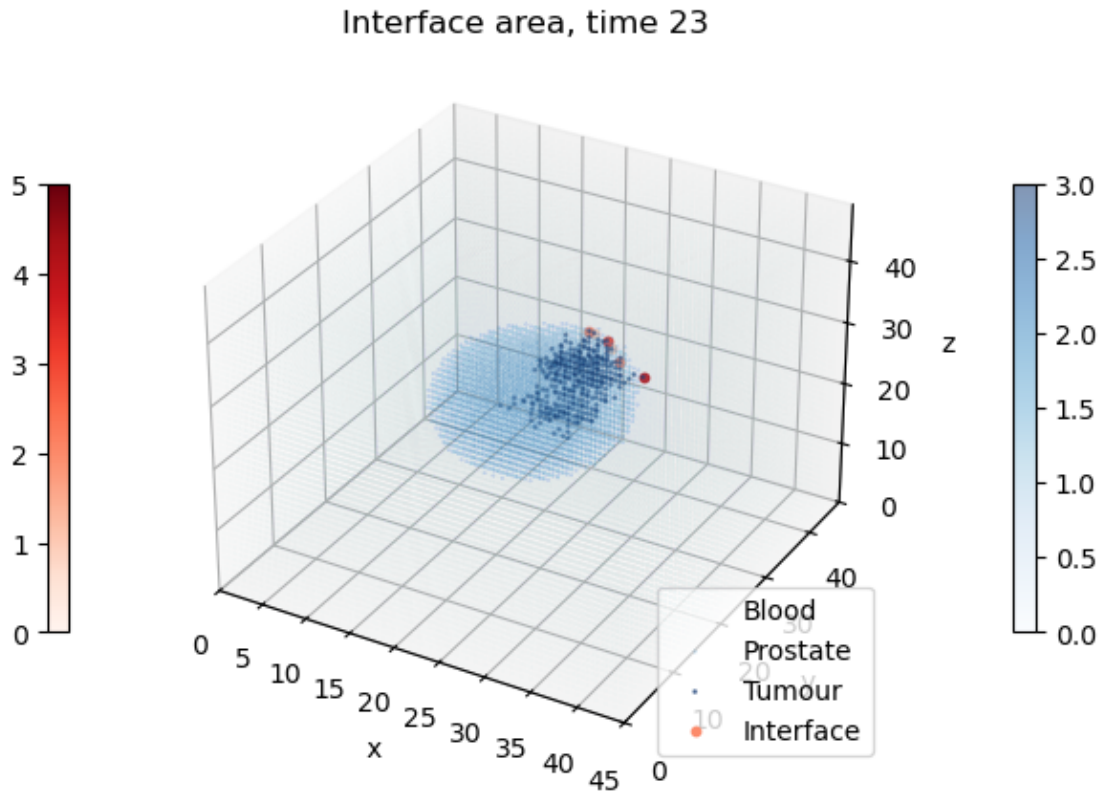

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z
0
10
20
30
40


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

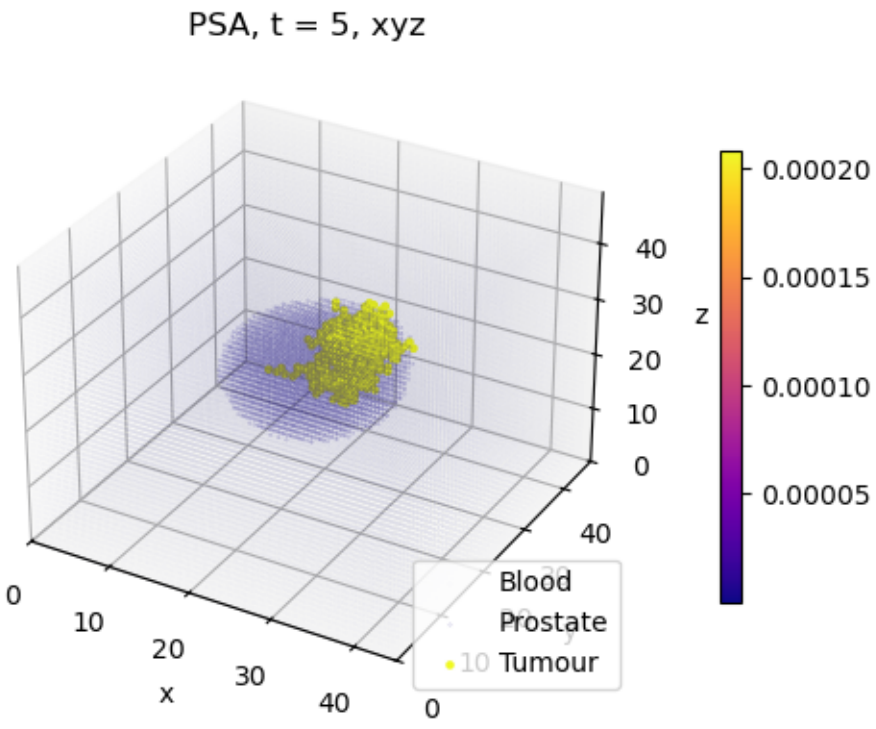

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

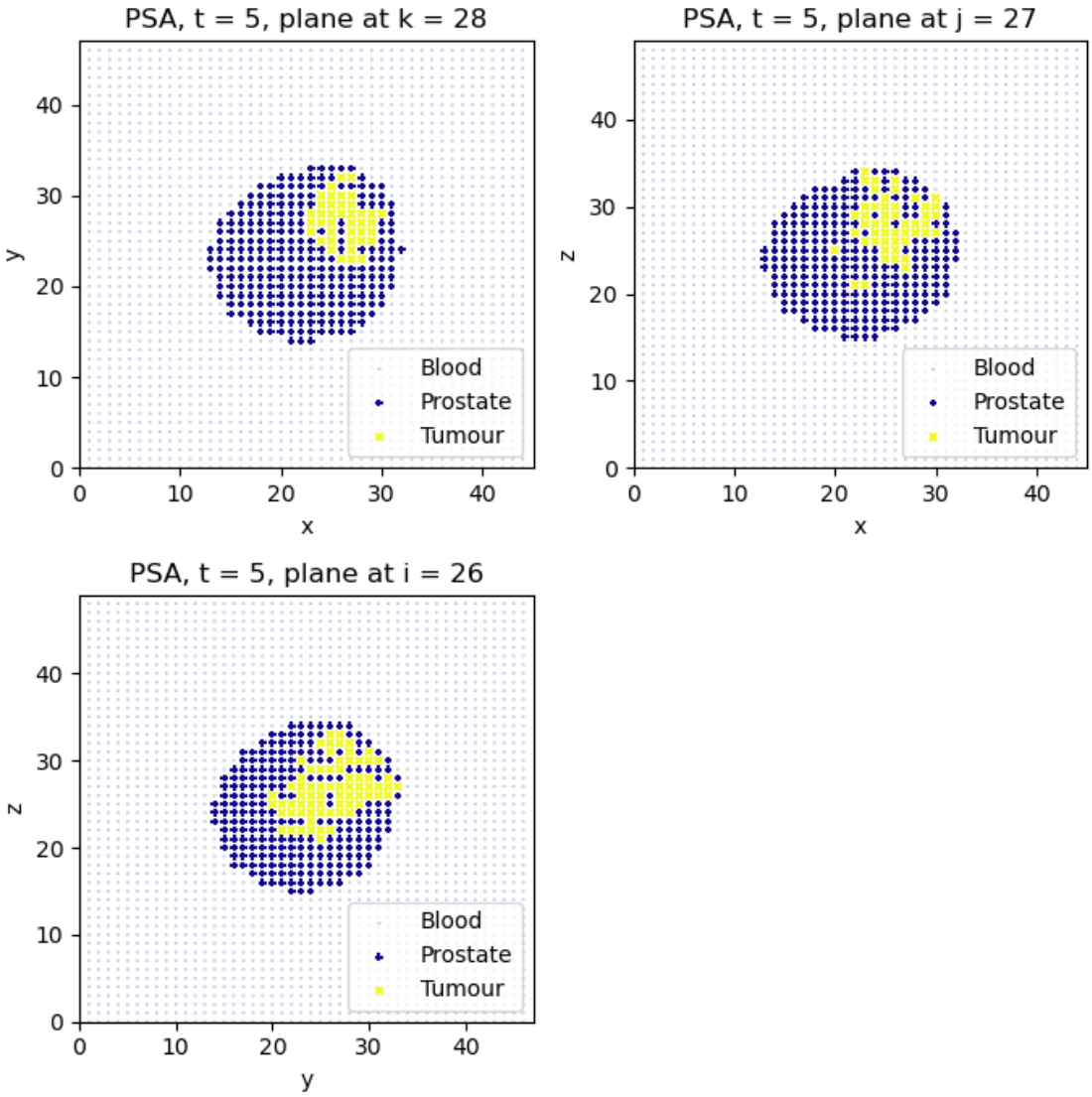

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3


PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

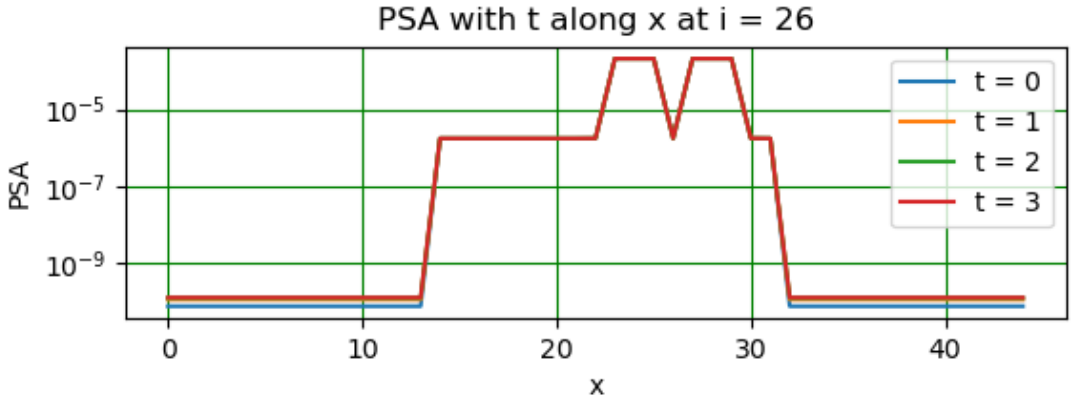


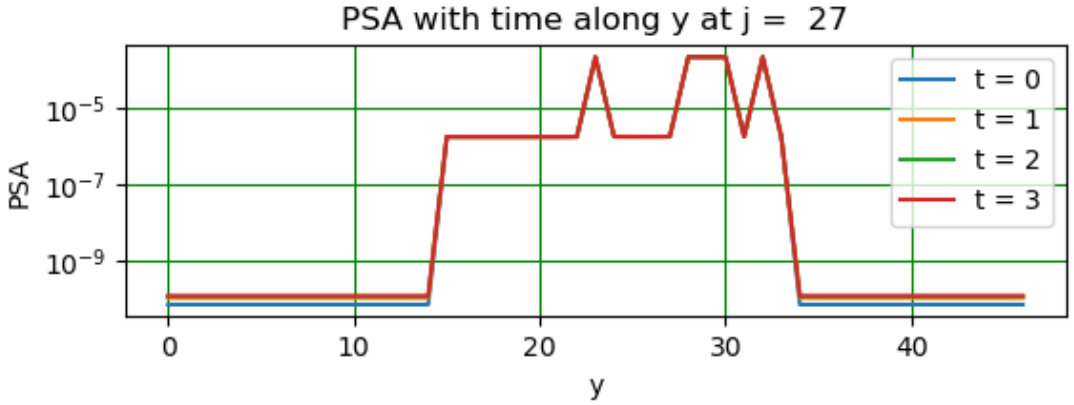

```
At time 23:
No. tumour cells 355, prostate cells 4138, blood cells 99142
PSA_top_level 7.468174e-11
--------------------------------------------------------------------------------
----------------------------
Time 24
Diffusion PSA_top, with tumour
Diffusion converged, nt = 5, ext_test = 8.410615e-04, prostate_test =
1.096618e-04
```


PSA, t = 5, xyz
0
10
20
30
40
x
0
10
20
30
40
z
0
10
20
30
40
0.00020
0.00015
0.00010
0.00005
Blood
Prostate
Tumour

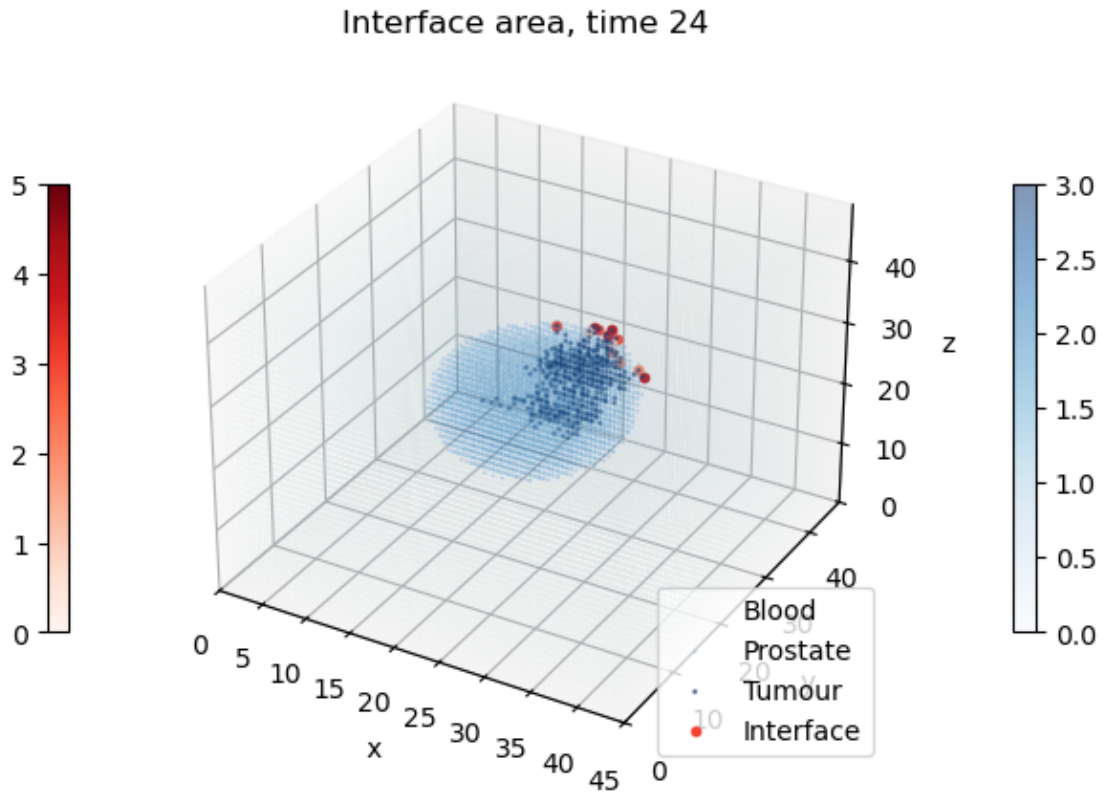

PSA, t = 5, plane at k = 28
PSA, t = 5, plane at j = 27
PSA, t = 5, plane at i = 26
Blood
Prostate
Tumour
x
y
z


PSA with t at specified points
log(PSA)
−15.0
−17.5
−20.0
−22.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
(26, 27, 28)
(22, 23, 24)
(0, 0, 0)
(1, 1, 1)
(11, 11, 12)

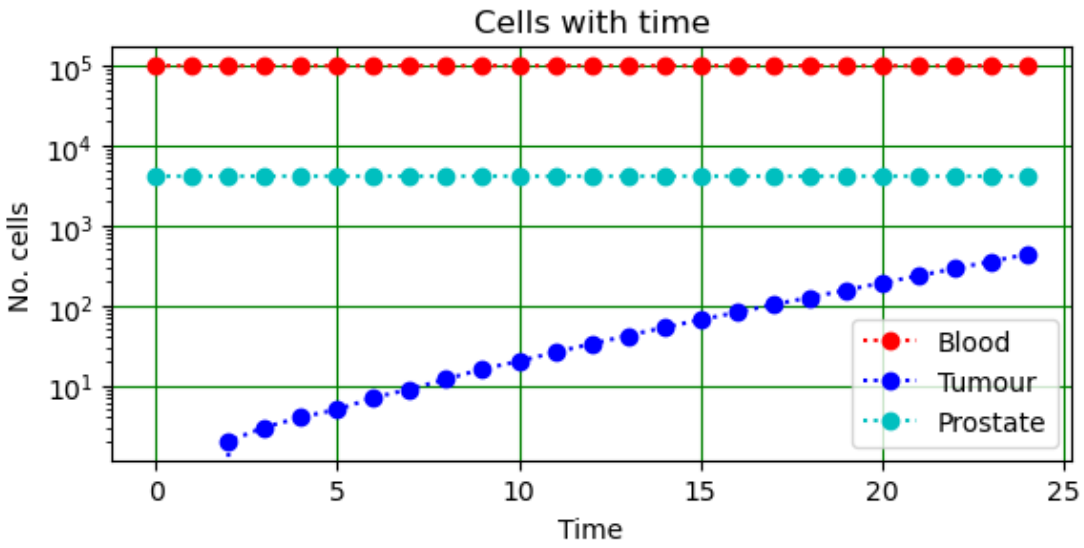

PSA with t along x at i = 26
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
x
t = 0
t = 1
t = 2
t = 3

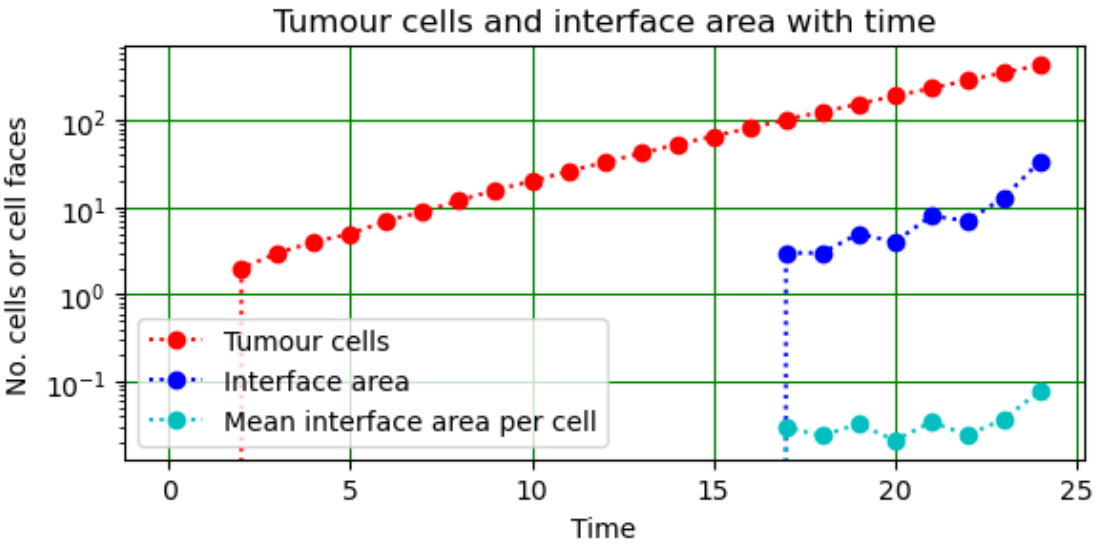

PSA with time along y at j = 27
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
y
t = 0
t = 1
t = 2
t = 3

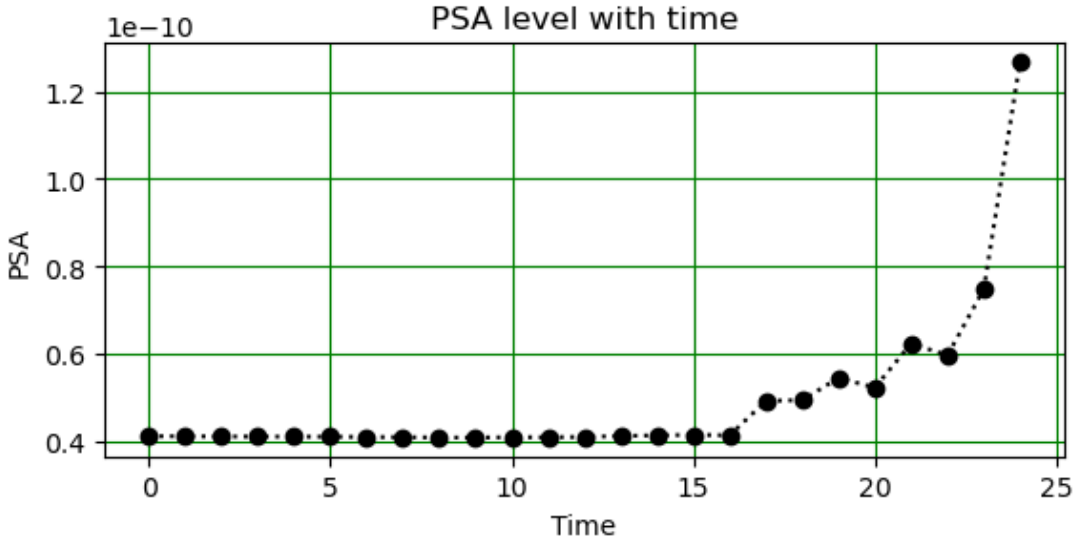

PSA with t along z at k = 28
PSA
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
z
t = 0
t = 1
t = 2
t = 3

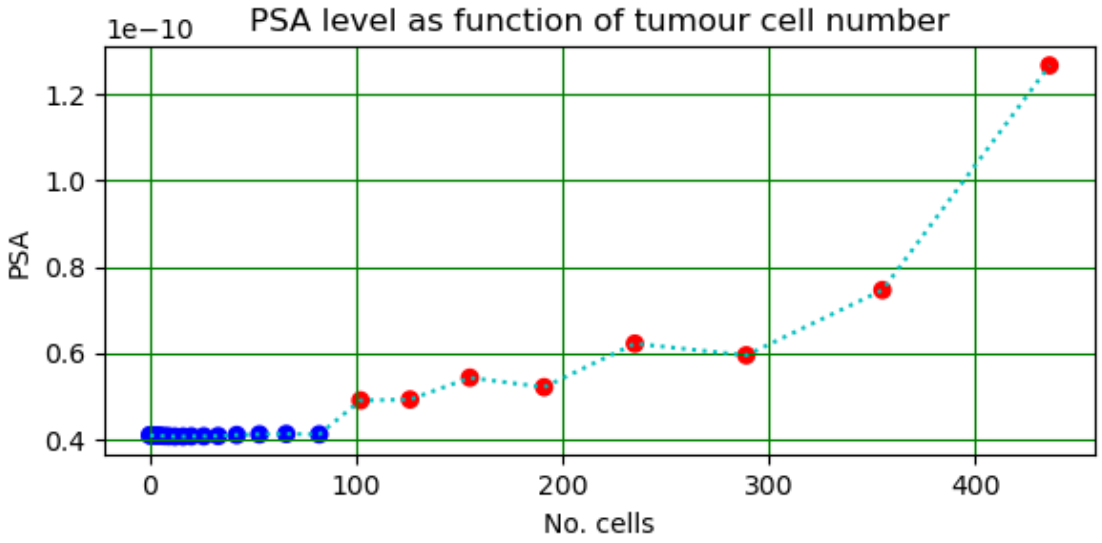


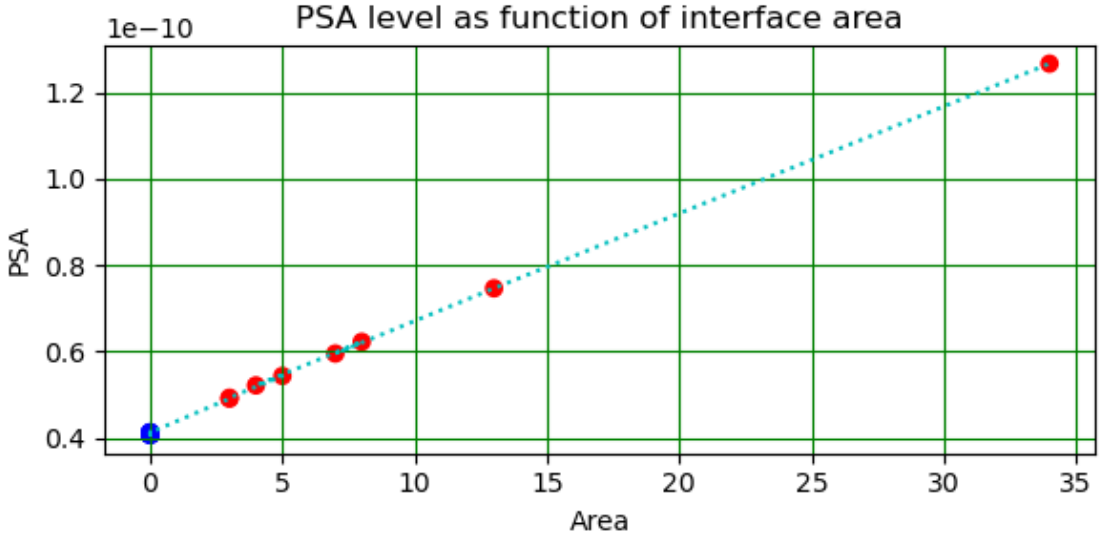

```
At time 24:
No. tumour cells 436, prostate cells 4138, blood cells 99061
PSA_top_level 1.267176e-10
```


Cells with time
No. cells
Time
Blood
Tumour
Prostate


Tumour cells and interface area with time
No. cells or cell faces
Time
Tumour cells
Interface area
Mean interface area per cell


PSA level with time
1e−10
PSA
Time






```
Date and time 2026-02-04 10:32:24.196625
Time since last check is 0:02:38.966899
```

1.8 End of model run

Return to ToC: »

1.9 Replot

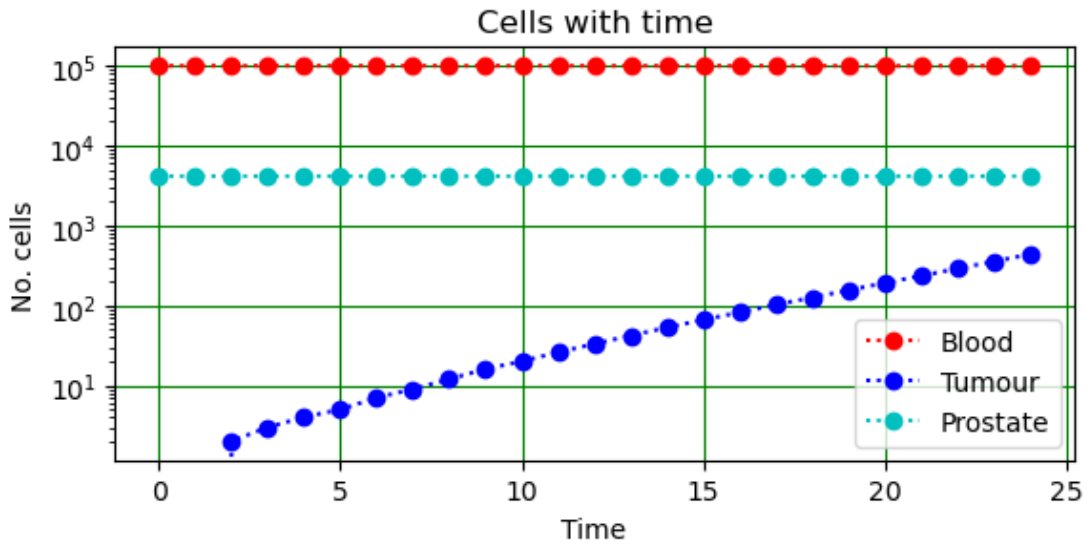
Return to ToC: »

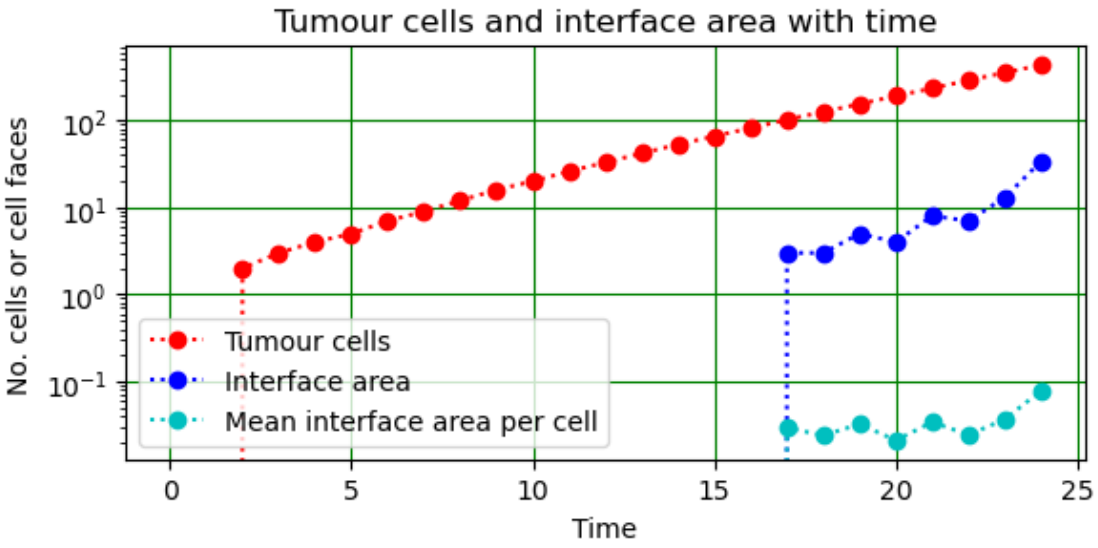
[17]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
#
print(" ")
print(f"Tumour centre, indices {i_tumour}, {j_tumour}, {k_tumour}")
print(f"Tumour radius {tumour_rad}")
print(f"Prostate centre, indices {i_prostate}, {j_prostate}, {k_prostate}")
print(f"Prostate radius {prostate_rad}")
#
fig = plt.figure(figsize = (6, 9))
#
ax = fig.add_subplot(3, 1, 1)
ax.set_title("Cells with time")
ax.set_xlabel("Time")
ax.set_ylabel("No. cells")
ax.plot(times, blood_time, marker = 'o', linestyle = ':', color = 'r', label =␣
 ↪"Blood")
ax.plot(times, tumour_time, marker = 'o', linestyle = ':', color = 'b', label =␣
 ↪"Tumour")
ax.plot(times, prostate_time, marker = 'o', linestyle = ':', color = 'c', label␣
 ↪= "Prostate")
ax.set_yscale('log')
ax.grid(color = 'g')
ax.legend()
#
ax = fig.add_subplot(3, 1, 2)
ax.set_title("Tumour cells and interface area with time")
ax.set_xlabel("Time")
ax.set_ylabel("No. cells or cell faces")
ax.plot(times, tumour_time, marker = 'o', linestyle = ':', color = 'r', label =␣
 ↪"Tumour cells")
ax.plot(times, area_time, marker = 'o', linestyle = ':', color = 'b', label =␣
 ↪"Interface area")
ax.plot(times, face_number, marker = 'o', linestyle = ':', color = 'c',
        label = "Mean interface area per cell")
ax.set_yscale('log')
ax.grid(color = 'g')
ax.legend()
#
ax = fig.add_subplot(3, 1, 3)
ax.set_title("PSA level with time")
ax.set_xlabel("Time")
ax.set_ylabel("PSA")
if double:
    ax.plot(times, PSA_top_time, marker = 'v', linestyle = '', color = 'k')
    ax.plot(times, PSA_bot_time, marker = '^', linestyle = '', color = 'k')
```

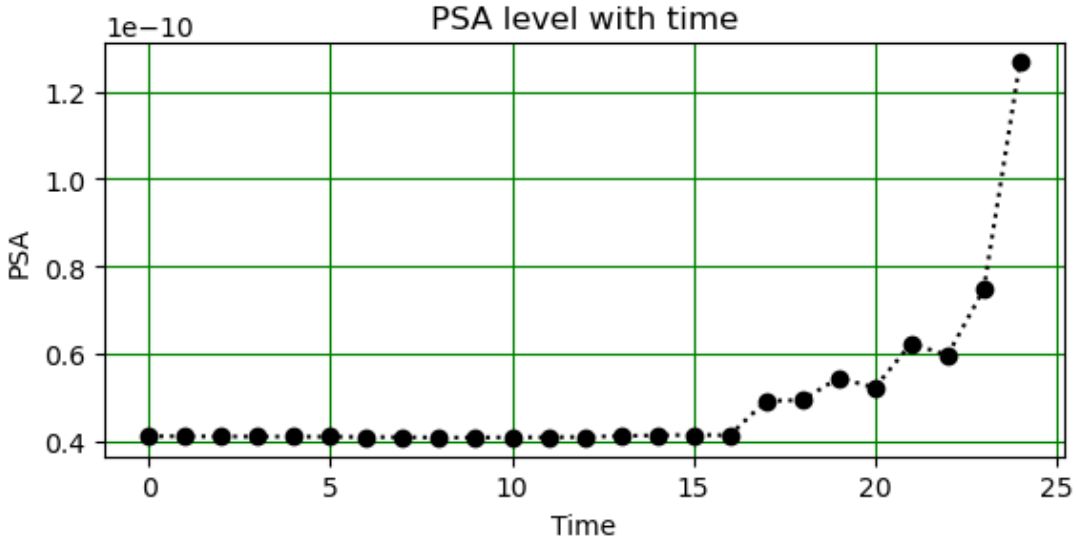
```
    ax.fill_between(times, PSA_bot_time, PSA_top_time, color = 'k', alpha = 0.3)
    if triple:
        ax.plot(times, PSA_mid_time, marker = '+', linestyle = '-', color = 'k')
else:
    ax.plot(times, PSA_top_time, marker = 'o', linestyle = ':', color = 'k')
ax.grid(color = 'g')
#
plt.tight_layout()
plt.show()
#
PSA_cells_time = tumour_time + prostate_time
#
plot_log_here = True
#
fig = plt.figure(figsize = (6, 6))
#
ax = fig.add_subplot(2, 1, 1)
ax.set_title("PSA level as function of tumour cell number")
ax.set_xlabel("No. cells")
ax.set_ylabel("PSA")
if double:
    ax.scatter(tumour_time, PSA_top_time, marker = 'v', c = btcell_color)
    ax.scatter(tumour_time, PSA_bot_time, marker = '^', c = btcell_color)
    ax.fill_between(tumour_time, PSA_bot_time, PSA_top_time, color = 'c', alpha␣
 ↪= 0.3)
    if triple:
        ax.scatter(tumour_time, PSA_mid_time, marker = '+', c = btcell_color)
else:
    ax.scatter(tumour_time, PSA_top_time, marker = 'o', c = btcell_color)
    ax.plot(tumour_time, PSA_top_time, marker = '', linestyle = ':', color =␣
 ↪'c')
ax.grid(color = 'g')
if plot_log_here and len(tumour_time > 0):
    ax.set_xscale('log')
    ax.set_yscale('log')
#
ax = fig.add_subplot(2, 1, 2)
ax.set_title("PSA level as function of interface area")
ax.set_xlabel("Area")
ax.set_ylabel("PSA")
if double:
    ax.scatter(area_time, PSA_top_time, marker = 'v', c = btcell_color)
    ax.scatter(area_time, PSA_bot_time, marker = '^', c = btcell_color)
    ax.fill_between(area_time, PSA_bot_time, PSA_top_time, color = 'c', alpha =␣
 ↪0.3)
    if triple:
        ax.scatter(area_time, PSA_mid_time, marker = '+', c = btcell_color)
```

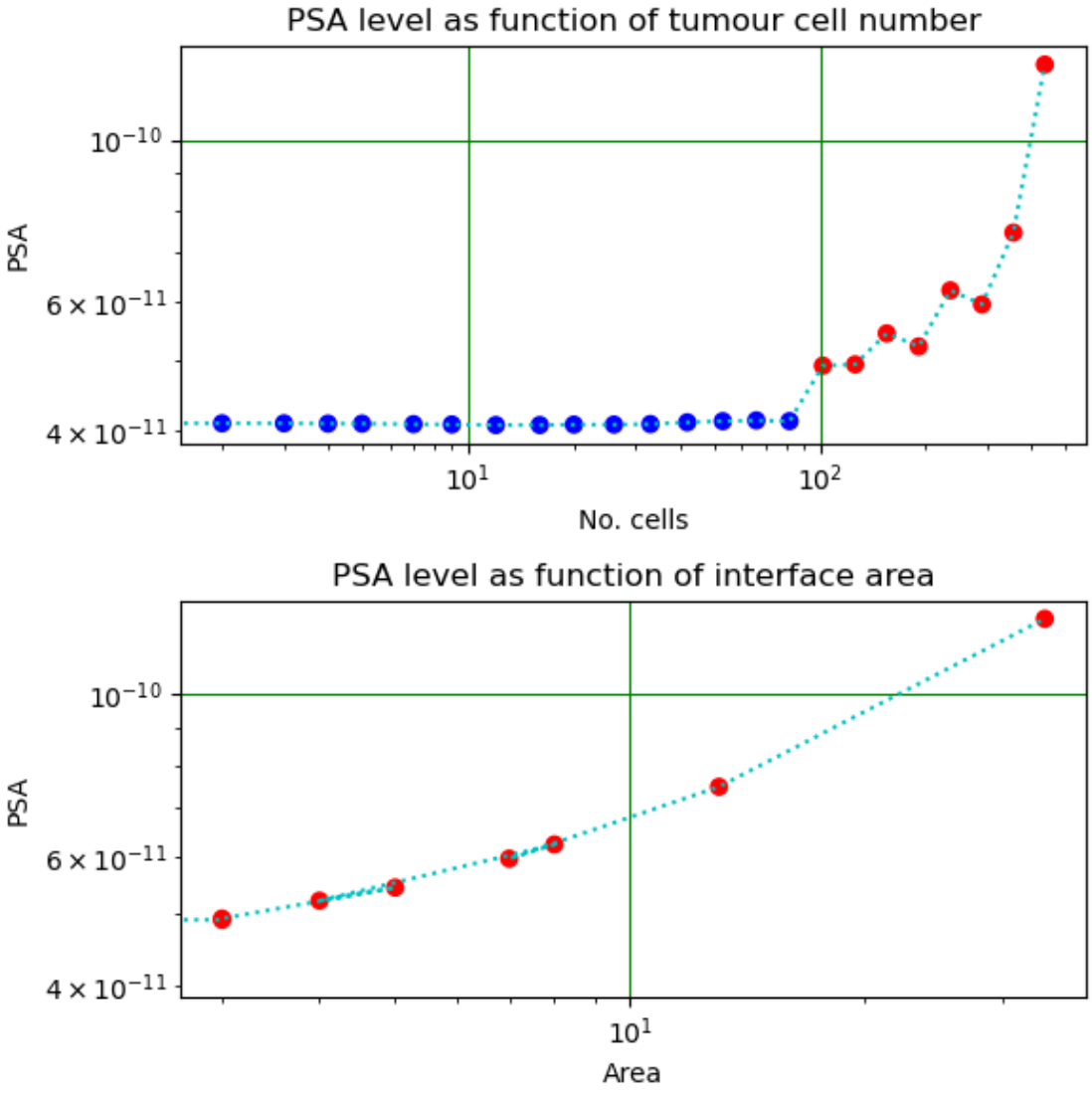
```
else:
    ax.scatter(area_time, PSA_top_time, marker = 'o', c = btcell_color)
    ax.plot(area_time, PSA_top_time, marker = '', linestyle = ':', color = 'c')
ax.grid(color = 'g')
if plot_log_here:
    ax.set_xscale('log')
    ax.set_yscale('log')
#
plt.tight_layout()
plt.show()
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:32:24.206859

Tumour centre, indices 26, 27, 28
Tumour radius 0.6
Prostate centre, indices 22, 23, 24
Prostate radius 10
```


Cells with time
No. cells
Time
Blood
Tumour
Prostate


Tumour cells and interface area with time
No. cells or cell faces
Time
Tumour cells
Interface area
Mean interface area per cell


PSA level with time
1e−10
PSA
Time




```
Date and time 2026-02-04 10:32:25.247112
Time since last check is 0:00:01.040253
```

1.10 Code cell template

Return to ToC: »

[18]:
```
import datetime
now = datetime.datetime.now()
print("Date and time",str(now))
```

```
#
#
then = now
now = datetime.datetime.now()
print("\nDate and time",str(now))
print("Time since last check is",str(now - then))
```

```
Date and time 2026-02-04 10:32:25.251290

Date and time 2026-02-04 10:32:25.251437
Time since last check is 0:00:00.000147
```

[ ]: