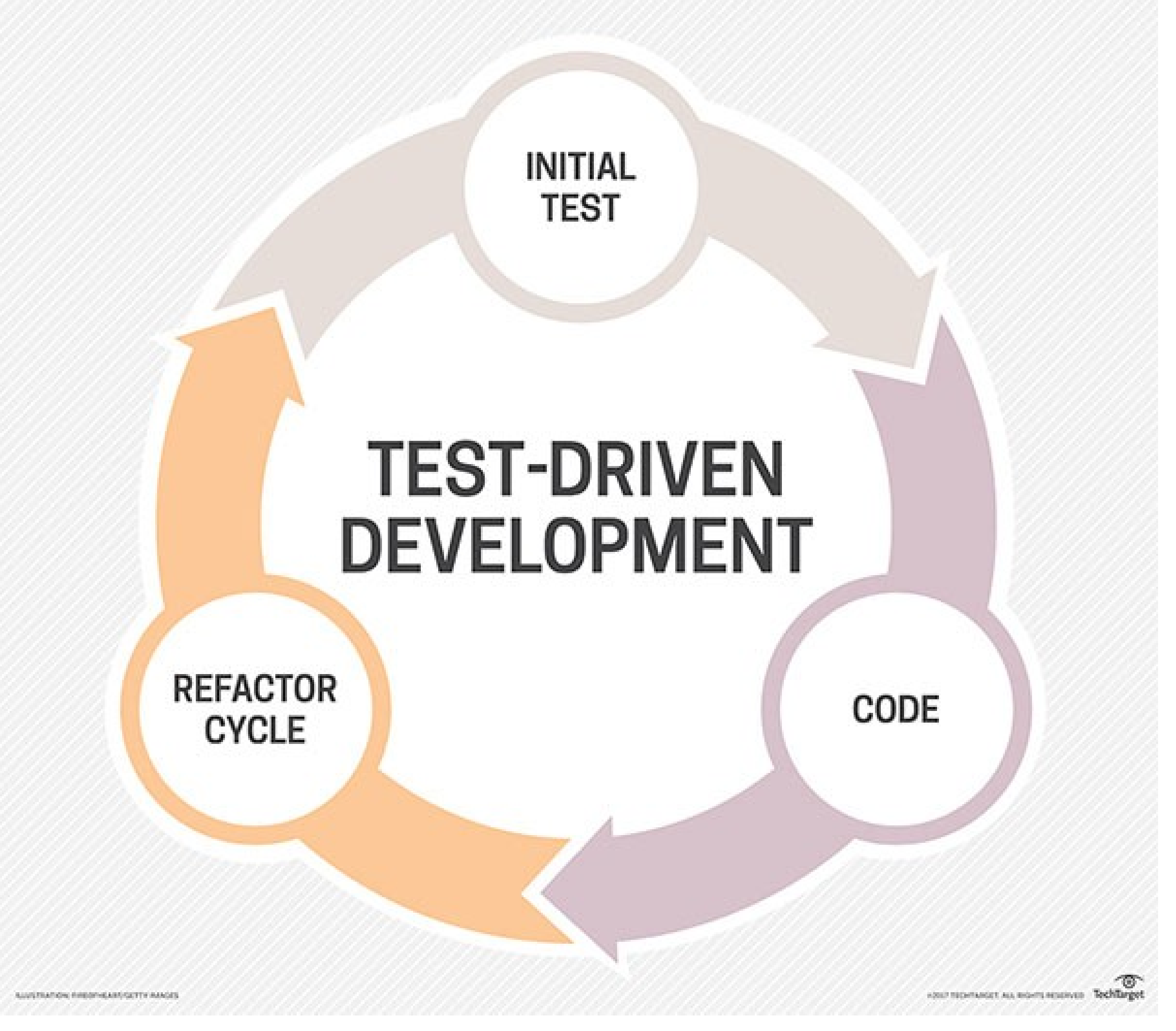
Test -Driven Development in Astronomy

James Nightingale

Credit:

Richard Hayes



The Astronomer's Development Cycle

The Astronomer’s Development Cycle

- **Step 1: Write Code.**

The Astronomer’s Development Cycle

- **Step 1: Write Code.**
- **Step 2: Write more code.**

The Astronomer’s Development Cycle

- **Step 1: Write Code.**
- **Step 2: Write more code.**
- **Step 3: Keep writing code, it’ll eventually work.**

The Astronomer’s Development Cycle

- **Step 1: Write Code.**
- **Step 2: Write more code.**
- **Step 3: Keep writing code, it’ll eventually work.**
- **Step 4: I think it does what its supposed to, I better not change it.**

The Astronomer’s Development Cycle

- **Step 1: Write Code.**
- **Step 2: Write more code.**
- **Step 3: Keep writing code, it’ll eventually work.**
- **Step 4: I think it does what its supposed to, I better not change it.**
- **Step 5 (6 months later): :(**

```fortran
! Allocate variables to store each pixels neighbours
allocate (CompPix(maxval(g_degree(:)+1)), dist_fast(maxval(g_degree(:)+1)) )

! Subpix counts which subpixel we are currently on
subpix = 0

!Use NN to allocate all subgridded pixels to nearest cluster centre
do I = 1, Image_Pix(ImNo)
   do J = 1, Src_isub(ImNo)**2

      ! The subpixels 'host' image / source pixel (the source pixel that was allocated to the sub pixels host image pixel)
      if ( (Src_Cluster_Sparse .eq. 'Off') ) then
         CompPix(1) = cluster_index(I)
      elseif ( Src_Cluster_Sparse .eq. 'On') then
         CompPix(1) = cluster_index_Sparse(Src_Sparse_Grid_IpPair(I,ImNo))
      end if

      ! Calculate the distance of this sub-pix to its 'host' source pixel ...
      subpix = subpix + 1
20    dist_fast(1) =  (Source_XY_isub_Arc(1,subpix,ImNo) - centers(1,CompPix(1)))**2 +  (Source_XY_isub_Arc(2,subpix,ImNo) - centers(2,CompPix(1)))**2

      ! ... and all of that source pixels neighbours
      do K = 2, g_degree(CompPix(1))+1
         if (g_neighbour(g_start(CompPix(1))+K-2) .gt. 0) then
            CompPix(K) = g_neighbour(g_start(CompPix(1)) + K-2)
            dist_fast(K) =  (Source_XY_isub_Arc(1,subpix,ImNo) - centers(1,CompPix(K)))**2 +  (Source_XY_isub_Arc(2,subpix,ImNo) - centers(2,CompPix(K)))**2
         else
            dist_fast(K) = 1.e8
         end if

      end do

      ! Find the sub-pixels closest source pixel
      list = CompPix(minloc(dist_fast(1:K-1)))

      ! If the closest source pixel was a neighbouring pixel and not its 'host' pixel, then we don't know this is its nearest neighbour.
      ! Therefore, set this new source pixel as its 'host' and redo the calc above, until the host is the closest
      if (CompPix(1) .ne. list(1)) then
         CompPix(1) = list(1)
         go to 20
      end if

      ! If the host was the closter, allocate in 'cluster_index_isub' and go on to next sub-pixel
      cluster_index_isub(subpix) = list(1)

   end do

   if ( Src_Cluster_Sparse .eq. 'On') then

21    dist_fast(1) =  (Source_XY_Arc(1,I,ImNo) - centers(1,CompPix(1)))**2 +  (Source_XY_Arc(2,I,ImNo) - centers(2,CompPix(1)))**2
```

The Astronomer’s Development Cycle

- **Structure and Planning is the difference between surgery...**

The Astronomer’s Development Cycle

- **Structure and Planning is the difference between surgery and cutting people’s bodies open.**

The Astronomer’s Development Cycle

- **Structure and Planning is the difference between surgery and cutting people’s bodies open.**
- **We as astronomers are not taught or encouraged to write code in a way that structure or planning.**
- **Enter – Test-Driven Development.**

The Test-Driven Development Cycle

The Test-Driven Development Cycle

- **Step 1: Write a unit test.**

code.py × test_code.py ×

```python
import pytest

import code

def test__add_integers__input_one_and_one__result_is_two():

    result = code.add_integers(1,1)

    assert result == 2
```

The Test-Driven Development Cycle

- **Step 1: Write a unit test.**
- **Step 2: Run the test, check it fails.**

```
==================================== FAILURES =====================================
_____________ test__add_integers__input_one_and_one__result_is_two _____________

    def test__add_integers__input_one_and_one__result_is_two():

>       result = code.add_integers(1,1)
E       AttributeError: module 'code' has no attribute 'add_integers'

test_code.py:7: AttributeError
=========================== 1 failed in 0.03 seconds ===========================
Process finished with exit code 0
```

The Test-Driven Development Cycle

- **Step 1: Write a unit test.**
- **Step 2: Run the test, check it fails.**
- **Step 3: Write the Code.**

code.py test_code.py

```python
def add_integers(integer_one, integer_two):
    return integer_one + integer_two
```

The Test-Driven Development Cycle

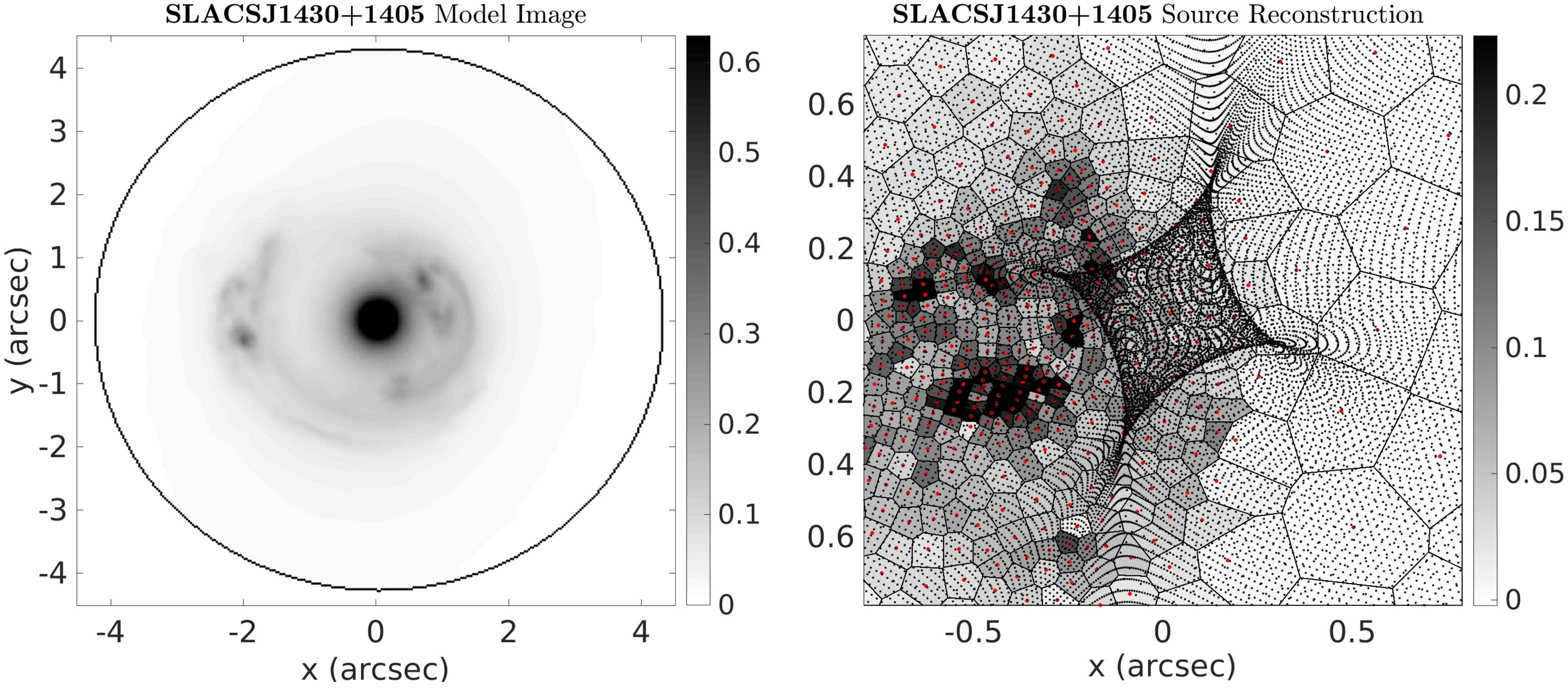
- **Step 1: Write a unit test.**
- **Step 2: Run the test, check it fails.**
- **Step 3: Write the Code.**
- **Step 4: Check the test (and all other tests) pass.**

1 test passed - 0ms

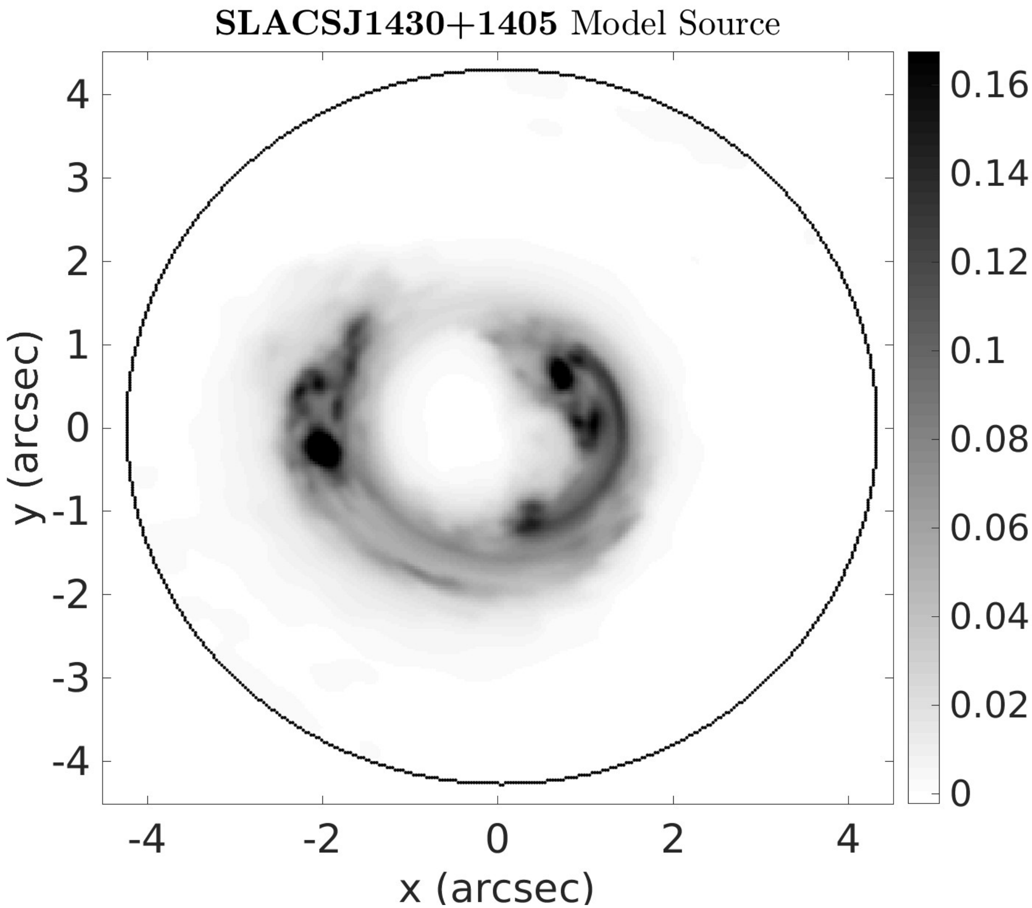
```
Testing started at 19:57 ...
/home/jammy/Euclid/VirtualEnvs/AutoLensPy3/bin/python /home/jammy/PyCharm/pycharm-community-2017.3.2/helpers/pycharm/_jb_pytest_runner.py --target test_code
Launching py.test with arguments test_code.py::test__add_integers__input_one_and_one__result_is_two in /home/jammy/PycharmProjects/TDDTalk

============================= test session starts ==============================
platform linux -- Python 3.6.3, pytest-3.4.2, py-1.5.2, pluggy-0.6.0
rootdir: /home/jammy/PycharmProjects/TDDTalk, inifile:
collected 1 item
test_code.py .                                                           [100%]

=========================== 1 passed in 0.01 seconds ===========================
Process finished with exit code 0
```

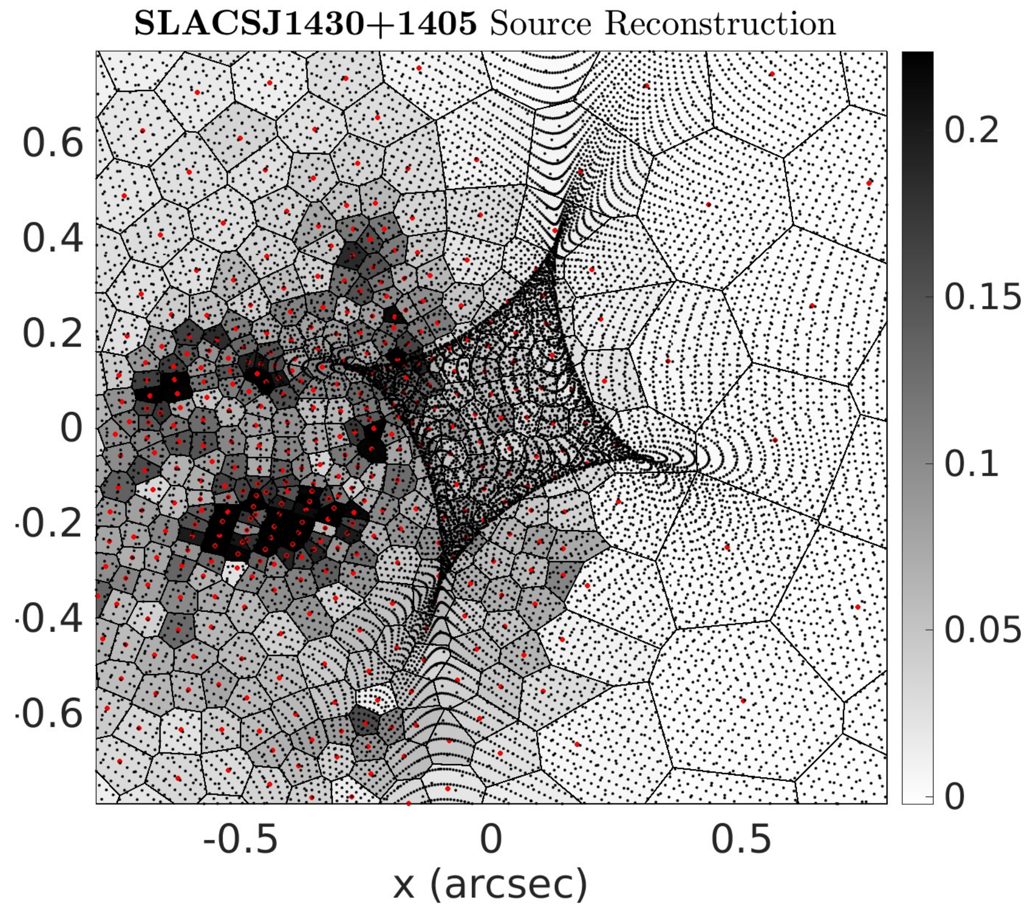
TDD – A (brief) case study

PyAutoLens – Open-source Strong Lens modeling



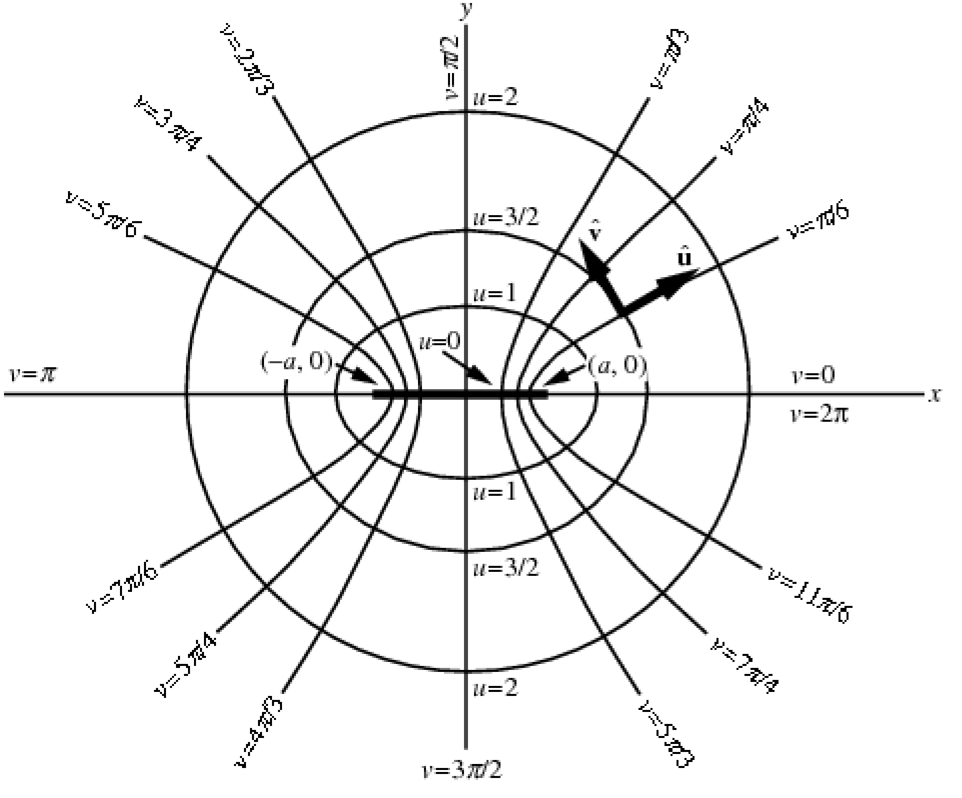
PyAutoLens – Open-source Strong Lens modeling





Coordinate Transform

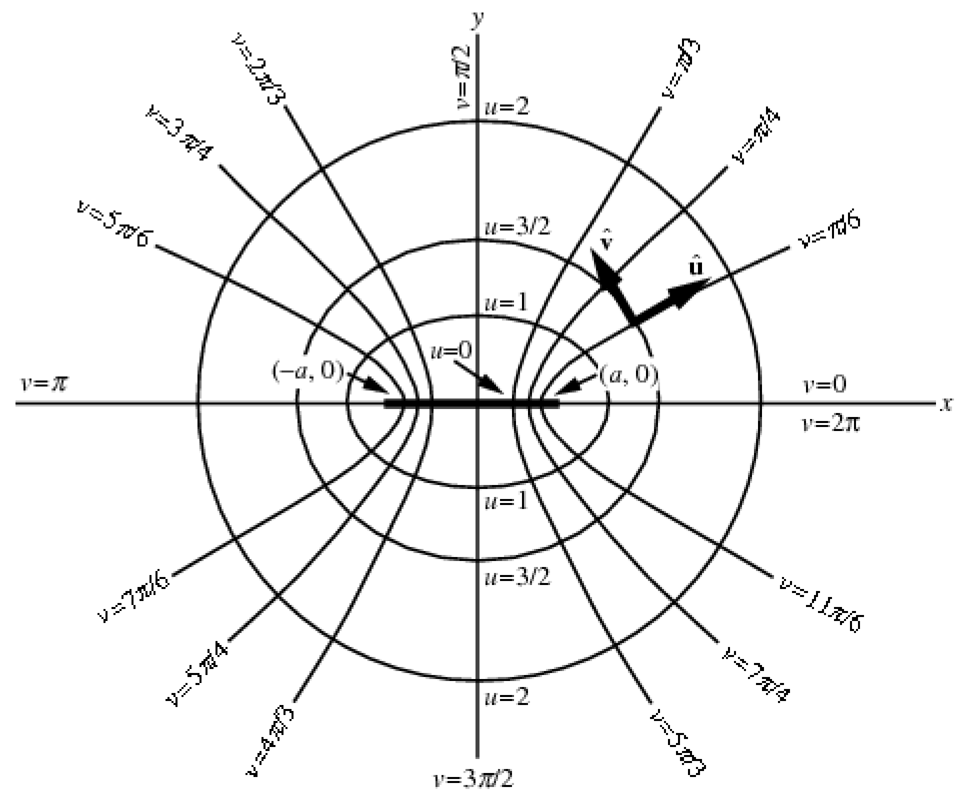
- **Image** – Cartesian Coordinates.
- **Galaxy** – Elliptical Coordinates.



TDD - Coordinate Transform

- **TDD forces you to break the problem down.**

  - I don't know how to write a unit test to perform this transformation.



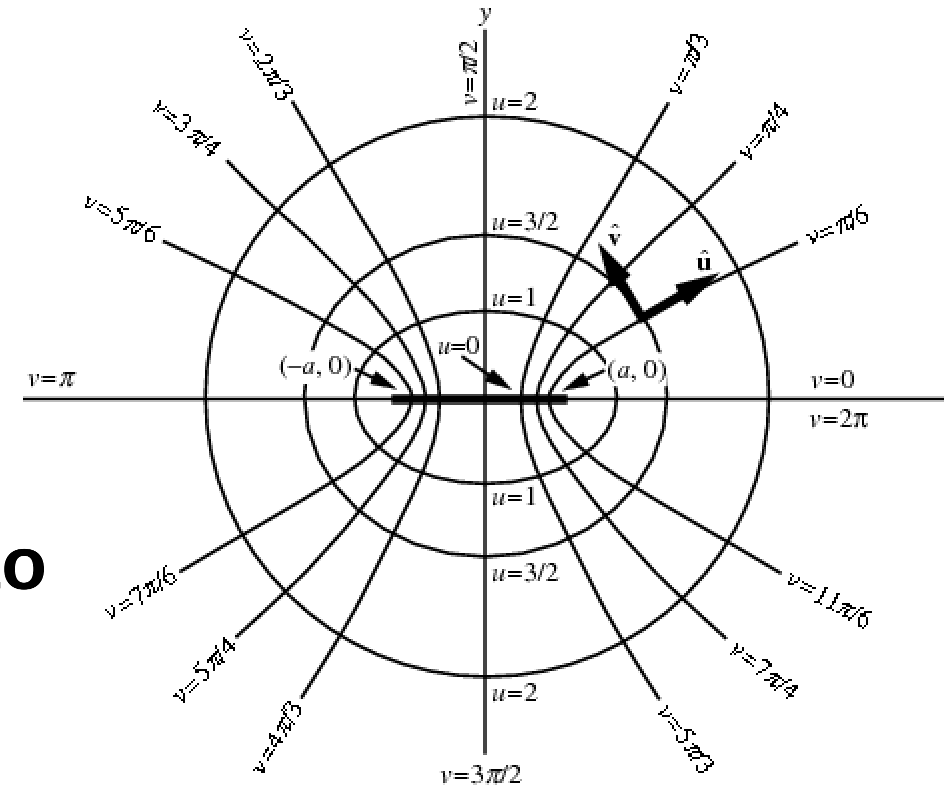
TDD - Coordinate Transform

- **TDD forces you to break the problem down.**

  - I don’t know how to write a unit test to perform this transformation.

  - However, I will need to **translate the coordinates to the galaxy’s centre.**

  - And that, **I know how to test.**



TDD - Coordinate Transform

```python
def test__coordinates_to_centre__input_coordinates_and_centre__shifts_coordinates_to_centre():

    coordinates_shift = geometry.coordinates_to_centre(coordinates=(0.0, 0.0), centre=(1.0, 1.0))

    assert coordinates_shift == (-1.0, -1.0)
```

TDD - Coordinate Transform

geometry.py × test_geometry.py ×

```python
def coordinates_to_centre(coordinates, centre):
    """
    Converts coordinates to a reference frame centre.

    Parameters
    ----------
    coordinates : (float, float)
        The x and y coordinates.
        coordinates : (float, float)
        The x and y centre.

    Returns
    ----------
    The coordinates at the reference frame's centre
    """
    return (coordinates[0] - centre[0], coordinates[1] - centre[1])
```

TDD - Coordinate Transform

geometry.py × test_geometry.py ×

```python
import pytest

import geometry

def test__coordinates_to_centre__input_coordinates_and_centre__shifts_coordinates_to_centre():

    coordinates_shift = geometry.coordinates_to_centre(coordinates=(0.0, 0.0), centre=(1.0, 1.0))

    assert coordinates_shift == (-1.0, -1.0)

def test__coordinates_to_centre__different_centre_and_coordinates():

    coordinates_shift = geometry.coordinates_to_centre(coordinates=(5.0, 2.0), centre=(4.0, 3.0))

    assert coordinates_shift == (1.0, -1.0)

def test__coordinates_to_centre__shift_only_x__only_x_shifts():

    coordinates_shift = geometry.coordinates_to_centre(coordinates=(0.0, 0.0), centre=(1.0, 0.0))

    assert coordinates_shift[1] == 0.0
    assert coordinates_shift == (-1.0, 0.0)

def test__coordinates_to_centre__shift_only_y__only_y_shifts():

    coordinates_shift = geometry.coordinates_to_centre(coordinates=(0.0, 0.0), centre=(0.0, 1.0))

    assert coordinates_shift[0] == 0.0
    assert coordinates_shift == (0.0, -1.0)
```

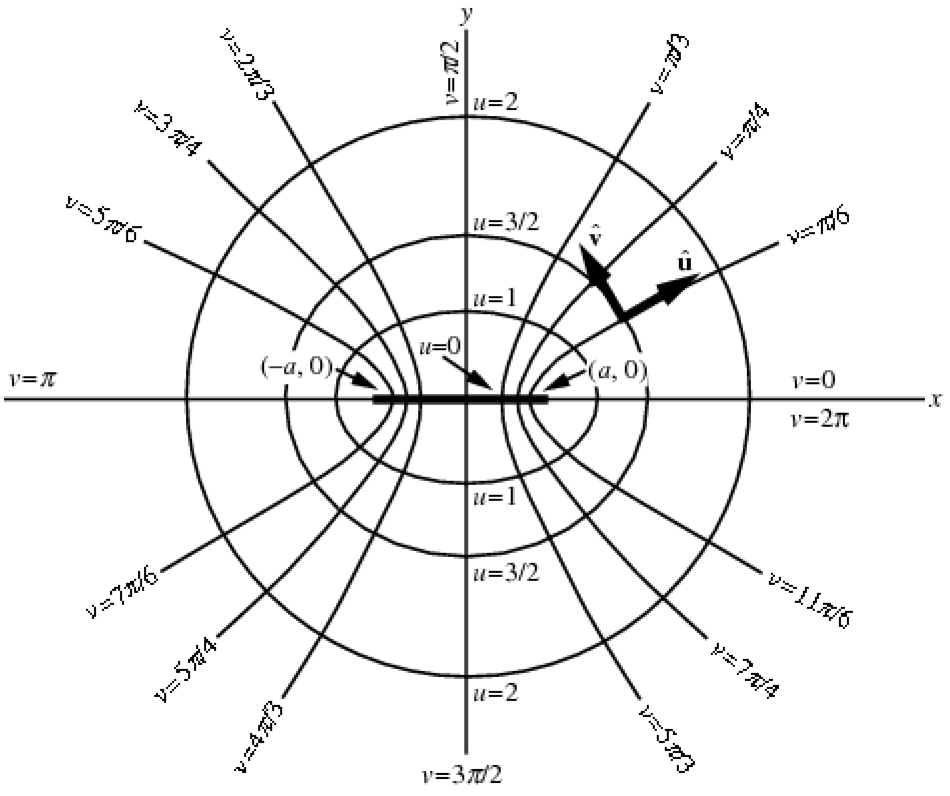
TDD - Coordinate Transform

- **TDD forces you to break the problem down.**

  - The angle between the coordinate and the x-axis.

  - The angle between the coordinate and galaxy.

  - Rotating the coordinate by that angle.



TDD - Coordinate Transform

geometry.py | test_geometry.py

```python
import numpy as np

def coordinates_to_centre(coordinates, centre):...

def coordinates_to_radius(coordinates):...

def coordinates_angle_from_x(coordinates):...

def coordinates_angle_to_galaxy(coordinates, theta, galaxy_theta):...

def rotate_coordinates_to_galaxy(cos_theta, sin_theta):...
```

TDD - Coordinate Transform

geometry.py × test_geometry.py ×

```
import geometry

def test__transform_to_galaxy_reference_frame__use_simple_functions():

    coordinates = (1.0, 1.0)
    centre = (2.0, 2.0)
    galaxy_theta = 45

    shifted_coordinates = geometry.coordinates_to_centre(coordinates, centre)

    radius = geometry.coordinates_to_radius(shifted_coordinates)

    theta_from_x = geometry.coordinates_angle_from_x(shifted_coordinates)

    cos_theta, sin_theta = geometry.coordinates_angle_to_galaxy(radius, theta_from_x, galaxy_theta)

    rotated_coordinates = geometry.rotate_coordinates_to_galaxy(cos_theta, sin_theta)

    assert rotated_coordinates == geometry.transform_to_galaxy_reference_frame(coordinates, centre, galaxy_theta)
```

TDD - Coordinate Transform

geometry.py × test_geometry.py ×

```python
import numpy as np

def coordinates_to_centre(coordinates, centre):...

def coordinates_to_radius(coordinates):...

def coordinates_angle_from_x(coordinates):...

def coordinates_angle_to_galaxy(coordinates, theta, galaxy_theta):...

def rotate_coordinates_to_galaxy(cos_theta, sin_theta):...

def transform_to_galaxy_reference_frame(coordinates, centre, galaxy_theta):
    """..."""

    shifted_coordinates = coordinates_to_centre(coordinates, centre)

    theta_from_x = coordinates_angle_from_x(shifted_coordinates)

    cos_theta, sin_theta = coordinates_angle_to_galaxy(shifted_coordinates, theta_from_x, galaxy_theta)

    return rotate_coordinates_to_galaxy(cos_theta, sin_theta)

```

TDD - Coordinate Transform

geometry.py | test_geometry.py

```python
import geometry

def test__transform_to_galaxy_reference_frame__use_simple_functions__():...

def test__transform_to_galaxy_reference_frame__x_aligned_with_galaxy__no_rotation__():...

def test__transform_to_galaxy_reference_frame__x_offset_180_degrees__coordinates_change_sign__():...

def test__transform_to_galaxy_reference_frame__answer_calculated_on_paper():...
```

TDD - Coordinate Transform

geometry.py | test_geometry.py

```python
import numpy as np

def coordinates_to_centre(coordinates, centre):...

def coordinates_to_radius(coordinates):...

def coordinates_angle_from_x(coordinates):...

def coordinates_angle_to_galaxy(coordinates, theta, galaxy_theta):...

def rotate_coordinates_to_galaxy(cos_theta, sin_theta):...

def transform_to_galaxy_reference_frame(coordinates, centre, galaxy_theta):
    """..."""

    shifted_coordinates = coordinates_to_centre(coordinates, centre)

    theta_from_x = coordinates_angle_from_x(shifted_coordinates)

    cos_theta, sin_theta = coordinates_angle_to_galaxy(shifted_coordinates, theta_from_x, galaxy_theta)

    return rotate_coordinates_to_galaxy(cos_theta, sin_theta)

```

Astronomer's Coordinate Transform

```
!     These common blocks pass other information to different model integrals
common /Int_coords_Arc/  npow, Xrot_Arc, Yrot_Arc
common /Int_eta/ eta

do I = 1, No_Defls

   R_Arc =  ((XYPos_Arc(1,I)-Lens_x_Arc)**2 + (XYPos_Arc(2,I)-Lens_y_Arc)**2)**0.5

   !     CAlculate cos theta / sin theta using trig (= x/r and y/r)
   costhe1=(XYPos_Arc(1,I)-Lens_x_Arc)/R_Arc
   sinthe1=(XYPos_Arc(2,I)-Lens_y_Arc)/R_Arc

   ! Perform rotation if ellpitical mass distribution
   dum=costhe1
   costhe=costhe1*Lens_cosphi+sinthe1*Lens_sinphi
   sinthe=sinthe1*Lens_cosphi-dum*Lens_sinphi

   ! Convert theta values to x and y using trig in rotated plane
   Xrot_Arc = R_Arc*costhe
   Yrot_Arc = R_Arc*sinthe

   !SIS Model is rotationally symmetric so treat seperately to avoid rotation

   If (Lens_Model_Defl .eq. trim('SIS')) then

      call Lens_Calc_Defl_Angles_SIS(XRot_Arc, YRot_Arc, Defl_Angles_Arc_Calc(:,I))

   elseif (Lens_Model_Defl .eq. trim('PtMass')) then

      call Lens_Calc_Defl_Angles_PtMass(costhe1, sinthe1, R_Arc, Defl_Angles_Arc_Calc(:,I))

   elseif ( Lens_Model_Defl .eq. trim('SIE')  ) then
```

TDD - Coordinate Transform … and More!

geometry.py × test_geometry.py ×

```python
class TransformedCoordinates(tuple):...


class CoordinatesException(Exception):...


class Profile(object):
    """Abstract Profile, describing an object with x, y cartesian image_grid"""

    def __init__(self, centre=(0.0, 0.0)):...

    # noinspection PyMethodMayBeStatic
    def transform_to_reference_frame(self, coordinates):...

    # noinspection PyMethodMayBeStatic
    def transform_from_reference_frame(self, coordinates):...

    def coordinates_to_centre(self, coordinates):...

    def coordinates_from_centre(self, coordinates):...

    def coordinates_to_radius(self, coordinates):...


class EllipticalProfile(Profile):...


class SphericalProfile(EllipticalProfile):
    """Generic circular profiles class to contain functions shared by light and mass profiles"""

    def __init__(self, centre=(0.0, 0.0)):
        """..."""
        super(SphericalProfile, self).__init__(centre, 1.0, 0.0)

```

TDD

Test-Driven Development

- TDD is NOT a **testing process.**
- The fact your code comes out fully tested is a **bonus.**

Test-Driven Development

- TDD Is a **development process.**

  - You focus on what the code should do, **before you write it.**

  - Leading to **versatile, clean and adaptable code.**

  - That has a **specified purpose.**

- **If you add unit-tests after writing the code, you are not doing TDD!**

Refactoring

The Test-Driven Development Cycle

- **Step 1: Write a unit test.**
- **Step 2: Run the test, check it fails.**
- **Step 3: Write the Code.**
- **Step 4: Check the test (and all other tests) pass.**
- **Step 5: Refactor, refactor and refactor.**

Refactoring

- In the Astronomers development cycle, **refactoring is terrifying.**

  \- You have **no idea** if your changes break the code.

  \- And even if you think they do, you cannot be **confident**.

Refactoring

- **With TDD, you receive instant feedback on if your code's functionality has changed.**

  - Refactoring becomes **enjoyable.**

  - You focus on **how to structure the code**, not **whether changing it will break it.**

- **The code design becomes part of the development cycle!**

Other TDD benefits

- The unit tests become **living, breathing documentation.**

  - They make the API of your code visible.

- For **collaborative projects,** TDD ensures other developers know what your code does.

  - And lets them know their changes don’t break it!

Summary

- **TDD is a development process that produces clean and verstile code.**
- **More astronomers should be using it!**
- https://github.com/Jammy2211/PyAutoLens
- Eposter – S11.05

TDD - Coordinate Transform

- At 225 degrees, a test failed!
- The trigonometry reverted back to -45 degrees.
- **TDD forced me to make a design choice about my code (and coordinate system) immediately.**
- **I’d have thought about this a lot later, one a lot more code was in place!**