Logbook

Ana Fabela Hinojosa ID: 27876594
PHS3360 project supervisor: Marcus Kitchen

"And a step backward, after making a wrong turn, is a step in the right direction."
— Kurt Vonnegut, "Player Piano"

Week -1 (12/07/21 –18/07/21)

I started reading the text suggested by Marcus Kitchen (**MK**) two weeks before the start of the semester
→ *Handbook of X-Ray Imaging: Physics and Technology*. 1 ed.

Aims:

1. Getting familiar with the **Transport-of-Intensity equation**
2. Create simulations of phase contrast imaging

Tasks:

1. Verify enrollment was successful
2. Formalise weekly meeting schedule
3. Get through Slides for each of the eight topics from **PHS4200/PHS4020** (X-ray optics)
4. Ask MK to link me to the lecture videos

15/07/21

Since my goals are so open at this stage; I think that a way to start me up in the topic is reading the materials that MK has sent me as thoroughly as possible and familiarise myself with the basics of the theory I'll be needing to create some simulations of phase contrast/retrieval imaging in my project.

X-ray Imaging theory fundamentals

(I include here some direct quotes and paraphrasing from Handbook of X-Ray Imaging: Physics and Technology. 1 ed., vol. 47, (CH.49))

Historical prelude

- W.C. Röntgen set out to search for phase effects in the X-ray beam (i.e. X-ray diffraction, refraction, and focusing), from his investigations he concluded that X-rays did not deviate by using ordinary optical elements.
- In 1899 Dutch physicists Haga and Wind found evidence of X-ray diffraction from a slit,
- X-ray diffraction from crystals was discovered and interpreted (Bragg and Bragg 1913; Friedrich et al. 1913; von Laue 1913).

Presently, given synchrotron and X-ray laboratory sources, **refraction** and **interference** phenomena of X-ray fields can be routinely observed and used for experiments.

Spectral decomposition of a wave function

The scalar field function, $\Psi(x,y,z,t)$ is the solution of the **d'Alembert equation** in a vacuum:

$$\left(\frac{1}{c^2}\frac{\partial^2}{\partial t^2} - \nabla^2\right)\Psi(x,y,z,t) = 0$$

where c is the speed of light in a vacuum, and $\nabla$is the Laplacian operator.
Using a **Fourier integral we spectrally decompose the scalar wavefunction**:

$$\Psi(x,y,z,t) = \frac{1}{\sqrt{2\pi}}\int_0^\infty \psi_\omega(x,y,z)e^{-i\omega t}d\omega,$$

Where each monochromatic component of the field with angular frequency ω, has been written as the product of a spatial part and a harmonic time-dependent term.

Using the Fourier integral above in the d'Alembert equation we obtain the **Helmholtz equation**!

$$\left(\nabla^2 + k^2\right)\psi_\omega(x,y,z) = 0$$

Deriving Helmholtz equation

$$\left(\frac{1}{c^2}\partial_t^2 - \nabla^2\right)\Psi(\vec{r},t) = 0$$

where $\Psi(\vec{r},t) = \frac{1}{\sqrt{2\pi}}\int_0^\infty \psi(\vec{r})\, e^{-i\omega t}\, d\omega$

$$\therefore \left(\frac{1}{c^2}\partial_t^2 - \nabla^2\right)\frac{1}{\sqrt{2\pi}}\int_0^\infty \psi(\vec{r})\, e^{-i\omega t}\, d\omega = 0$$

$$\therefore \int_0^\infty \left(\frac{1}{\sqrt{2\pi}}\frac{1}{c^2}\partial_t^2\psi(\vec{r})\, e^{-i\omega t}\, d\omega - \frac{1}{\sqrt{2\pi}}\nabla^2\psi(\vec{r})\, e^{-i\omega t}\, d\omega\right) = 0$$

$$\therefore \int_0^\infty \left(\frac{1}{\sqrt{2\pi}}\frac{1}{c^2}\psi(\vec{r})(-i\omega)^2 e^{-i\omega t}\, d\omega - \frac{1}{\sqrt{2\pi}}e^{-i\omega t}\nabla^2\psi(\vec{r})\, d\omega\right) = 0$$

$$\therefore \frac{-1}{\sqrt{2\pi}}\int_0^\infty \left(\frac{\omega^2}{c^2}\psi(\vec{r})\, e^{-i\omega t}\, d\omega + e^{-i\omega t}\nabla^2\psi(\vec{r})\, d\omega\right) = 0$$

$$\therefore \frac{-1}{\sqrt{2\pi}}\int_0^\infty \left(k^2\psi(\vec{r})\, e^{-i\omega t} + \nabla^2\psi(\vec{r})\, e^{-i\omega t}\right) d\omega = 0$$

$$\therefore \left(k^2 + \nabla^2\right)\psi(\vec{r})\int_0^\infty \frac{1}{\sqrt{2\pi}}e^{-i\omega t}\, d\omega = 0$$

$\mathcal{F}\{\delta(\omega)\}$? NO IDEA!

$$\therefore \left(k^2 + \nabla^2\right)\psi(\vec{r}) = 0$$

where $k = \omega/c$ (valid in the vacuum).

Describing an imaging process by means of the Helmholtz equation usually suffices when the X-ray beam is monochromatic (or quasi monochromatic), a situation commonly encountered in synchrotron-based X-ray imaging. However, most of the radiological imaging is done with polychromatic X-ray generated by X-ray tubes with non-trivial spectral properties. In this case the spectral decomposition of the wave function into monochromatic components has a huge practical importance: it enables the description of imaging processes with a time-independent approach for each monochromatic component, which can then be combined to obtain the complete description of the polychromatic process. (Pelliccia et al. 972)

16/07/21

Paraxial fields

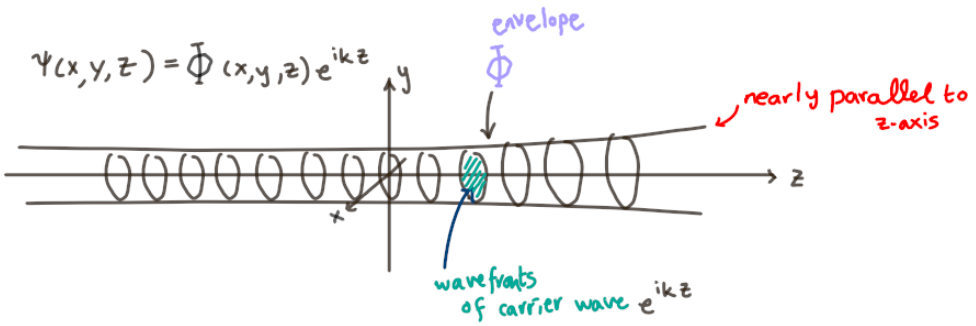
A **beam** is an EM wave propagating in free space with small divergence.
Electromagnetic energy in this context is concentrated within a small region about the beam axis, with small spread , this is called a **paraxial field**.
Paraxial == parallel to the beam axis. In most situations involving **phase-contrast imaging**, X-ray fields behave as paraxial fields.

Mathematical Formalism

Using the monochromatic spatial scalar wavefunction $\psi_\omega(x,y,z)$ solution to the Helmholtz equation as a paraxial field when

$$\psi_\omega(x,y,z) = \Phi(x,y,z)e^{ikz},$$

with smaller variations in the longitudinal direction **z** in contrast to the transverse **(x,y)-plane**.

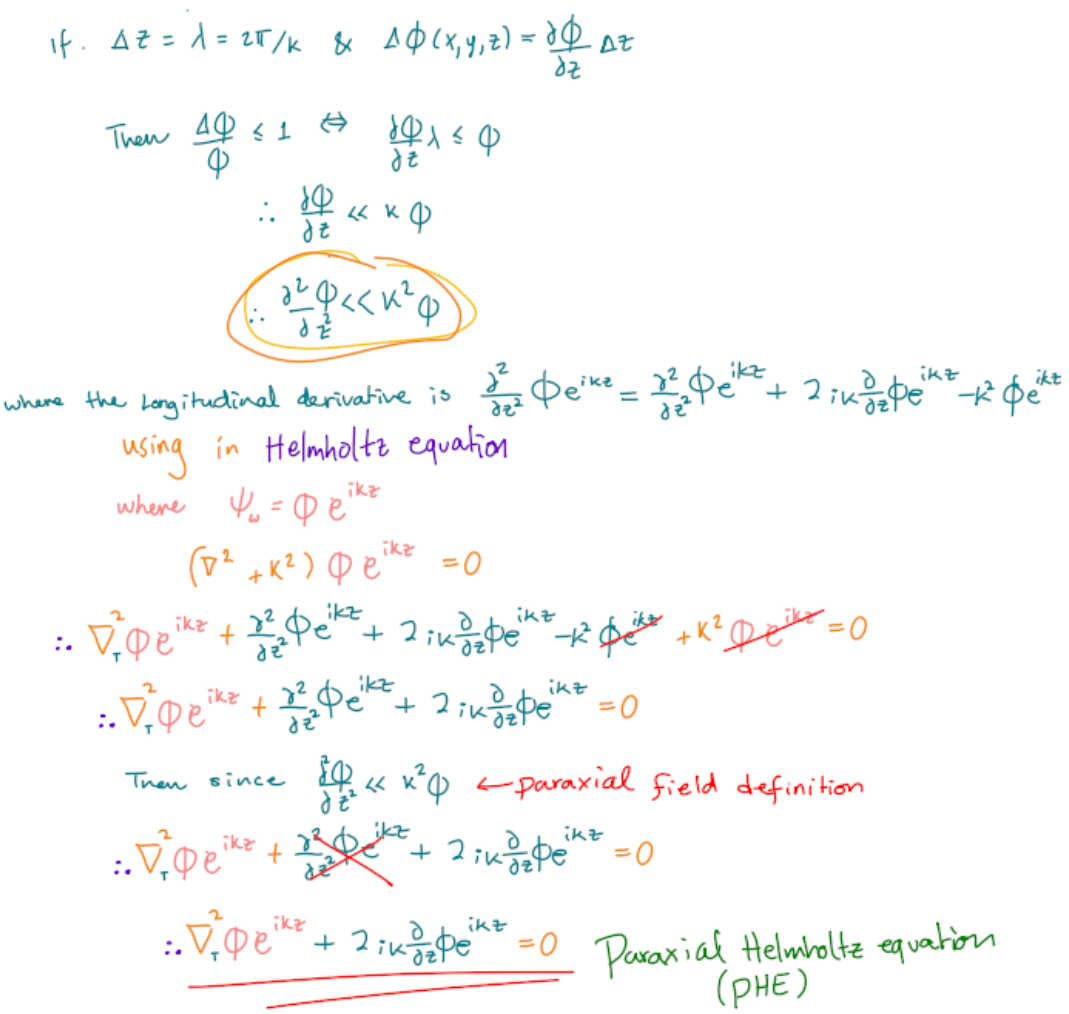


$\psi_\omega(x,y,z)$ displays beam propagation properties along the z-axis. The slowly varying complex envelope $\Phi(x,y,z)$ is modulated by a carrying pane wave e^{ikz}

The variation of the complex envelope within a **longitudinal distance** $\Delta z = \lambda = 2\pi/k$ must be smaller than the complex envelope itself

$$\frac{\Delta\Phi(x,y,z)}{\Phi(x,y,z)} \leqslant 1$$

,



The PHE is the approximation of the **homogeneous Helmholtz equation** under the condition of a slowly varying envelope.

Important points

Notable solutions to this equation are **Gaussian beams and Parabolic wavefronts**

Week 0 (19/07/21 –25/07/21)

Aims:

1. Understand the mathematical tools of
    a. **X-ray phase-contrast** effects in imaging
    b. methods to extract quantitative information from phase-contrast data.
2. Create simulations of phase contrast imaging

Tasks:

1. Getting familiar with the **Transport-of-Intensity equation**
2. ~~Verify enrollment was successful~~
3. Formalise weekly meeting schedule. Our first meeting will occur on **27th july** (TBC)
4. **Read X-Ray Handbook**
5. ~~Ask MK to link me to the lecture videos from **PHS4200/PHS4020** (X-ray optics)~~
6. Get through Slides for each of the topics from **PHS4200/PHS4020**

Interaction mechanisms of X-rays with matter

Inhomogeneous equations a quick revision

If **not** all the terms in the equation contain the **dependent variable** or its **partial derivatives,**
Then the equation is **inhomogeneous**
Examples:

1. $$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} = 0$$
2. $$\frac{\partial u}{\partial t} + x\frac{\partial^2 u}{\partial x^2} = x\exp(-t)$$

Refractive index

The **refractive index** $(n(x,y,z))$ is a position-dependent, frequency dependent () we omit this symbol from the notation) and time-independent quantity defined as

$$n(x,y,z) = \left(\frac{\varepsilon(x,y,z)}{\varepsilon_0}\right)^{1/2},$$

Where $\varepsilon(x,y,z)$ is the electrical **permittivity** of the material and ε_0 is the **permittivity** of the vacuum.

Note: In the presence of a material the **d'Alembert equation** is multiplied by the **refractive index** of the material in question.

Note: All equations presented so far are valid for a scalar field (i.e. a single component of the magnetic field)

In general, the wave function is a vector quantity, describing the evolution of both electric and magnetic field components and their mixing….Nevertheless, for most cases in hard X-ray imaging, the variations in the optical density of any media are sufficiently slowly varying over length scales comparable to the wavelength, that the electric and magnetic field components can be considered effectively decoupled (Pelliccia et al. 974)

In the presence of media, the complex field obeying the d'Alembert equation can still be written as a **superposition of monochromatic fields by** using the Fourier integral to spectrally decompose the scalar wave function. Hence each monochromatic component obeys the **inhomogeneous Helmholtz equation**.

$$(\nabla^2 + n^2(x,y,z)k^2)\psi(x,y,z) = 0.$$

Since paraxiallity can be assumed to be valid almost always we write the **inhomogeneous PHE** as

$$\left(\nabla_T^2 + 2ik\frac{\partial}{\partial z} + (n^2(x,y,z) - 1)k^2\right)\Phi(x,y,z) = 0$$

The refractive index has peculiar characteristics in the X-ray region of the spectrum that can be used to simplify the solution of the inhomogeneous equation (Pelliccia et al. 974).

Refractive index values	vacuum	Optical materials
Visible spectrum	$n(x,y,z) = 1$	$n(x,y,z) > 1$
X-Rays		$n(x,y,z) = 1 - \delta + i\beta$

Where δ is related to the **refraction** (**dispersion**), and β to the **absorption**, $\delta, \beta << 1$

The deviation from unity in the case of X-rays and optical materials is very small 10^{-6} **which means we can approximate the refractive behaviour of X-rays as null.**

Conventional radiology is based on **attenuation imaging**, related to the imaginary part of the complex refractive index. **Phase-contrast,** on the other hand, yields a quantity that is related to the real part, δ.

Phase-contrast *is a method of imaging that has a range of different applications. It exploits differences in the refractive index of different materials to differentiate between structures under analysis.* (Wikipedia, the free encyclopedia)

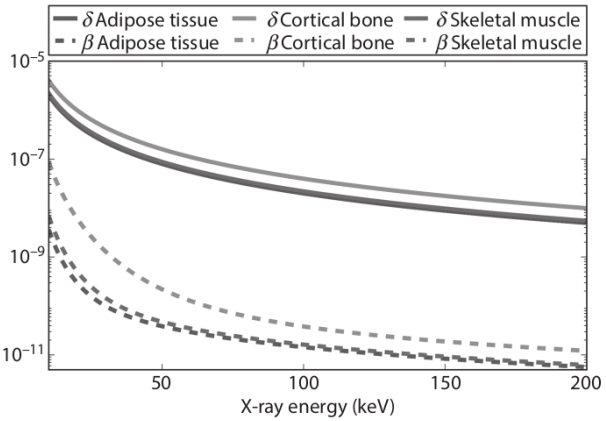


FIGURE 49.2 Calculated values of the refractive index for three representative human tissues: adipose tissue, cortical bone, and skeletal muscle. In all cases the real part, δ, of the refractive index is plotted with solid lines, and the imaginary part, β, is plotted with dashed lines.

Screenshot of figure 49.2 in Handbook of X-ray imaging (Pelliccia et al. 974).

From the figure above we can see that a very interesting comparison can be made when considering the **difference** in contrast between similar tissues.
The difference between adipose and skeletal muscle tissue is extremely small.

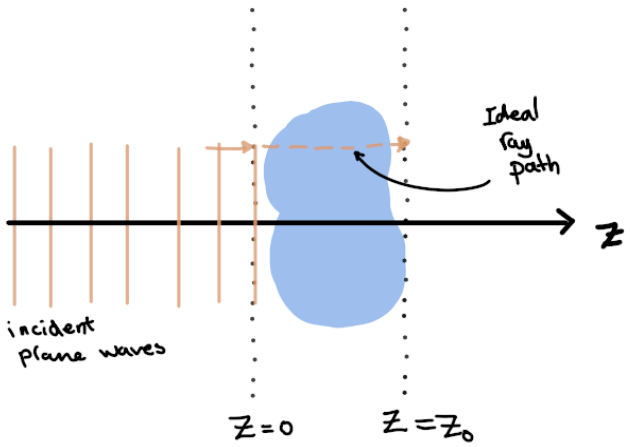
Comparing these values, i.e. $\Delta\delta = \delta_{mus} - \delta_{adip}$ vs $\Delta\beta = \beta_{mus} - \beta_{adip}$ we can conclude that $\Delta\delta >> \Delta\beta$.

This means that phase-contrast methods enable tiny differences in the real part of the refractive index to be more readily measured over attenuation contrast. In other words, **phase-contrast may offer an advantage when one needs to distinguish between different types of soft tissues** (Pelliccia et al. 974).

The projection approximation

This approximation describes the passage of rays through an object, by defining a nominal exit-surface, immediately "downstream" of an irradiated object, at which transverse phase and intensity changes are imprinted. The projection approximation assumes that all scattering within the object is fully described by this exit wave, with negligible diffraction within the scattering volume. (Pelliccia et al. 975)

Most X-rays passing through an object do not interact with the object at all. This is due to the small magnitude of the complex refractive index. The resulting imaging contrast is quite poor compared to other forms of imaging. **The weakness of X-ray interactions with objects is what yields the projection approximation**.



This drawing is a copy of figure 49.4 in the handbook. The projection approximation shows the ideal path traversed by the ray if there were no sample in the path of the incident plane waves. Scattering within the sample is considered negligible.

The scattering within the sample can be described by the exit wave defined at $z = z_0$.
Transverse changes in the amplitude and phase, accumulated by the unscattered wavefront when traversing the sample, are imprinted in the exit wave.
Neglecting scattering effects is mathematically equivalent to discarding the transverse laplacian term in the inhomogeneous Helmholtz equation:

$$(\nabla_T^2 + 2ik\frac{\partial\Phi(x,y,z)}{\partial z} + k^2(n^2(x,y,z) - 1)\Phi(x,y)) = 0.$$

Therefore, in the projection approximation we have the **inhomogeneous paraxial Helmholtz equation**

$$2ik\frac{\partial\Phi(x,y,z)}{\partial z} + (n^2(x,y,z) - 1)\Phi(x,y,z) = 0$$

*****Am I missing a factor of k in the second term? Is this a typo in the notes?***** yes!!! The factor of k should be present in the second term

20/07/21

Attenuation, phase and refraction of X-ray phase-contrast

The solution to the projection approximation evaluates the wave field at position $z = z_0$

$$\Phi(x,y,z_0) = \exp\left[\frac{k}{2i}\int_0^{z_0}(1 - n^2(x,y,z))dz\right]\Phi(x,y,0).$$

Because the refractive index is complex

$$n(x,y,z) = 1 - \delta + i\beta \quad , \quad \delta,\beta << 1,$$

$$\therefore \; n^2(x,y,z) \approx 1 - 2\delta + 2i\beta + 2i\delta\beta + \delta^2 + \beta^2$$

$$\therefore n^2(x,y,z) \approx 1 - 2\delta + 2i\beta$$

Therefore the **projection approximation** is,

$$\Phi(x,y,z_0) = \exp\left[-ik\int_0^{z_0}(\delta(x,y,z) - i\beta(z,y,z))dz\right]\Phi(x,y,0).$$

The exponential term is aka an **OPERATOR**

This formula is an expression for the **phase shift and attenuation** undergone by an X-ray wave moving across a sample in the projection approximation

Phase shift (complex component)	Attenuation
$\Delta\phi = -k\int_0^{z_0}\delta(x,y,z)dz$	$-k\int_0^{z_0}\beta(x,y,z)dz$

Assuming we have a homogeneous material can help getting a better understanding of the physics.

The (single, homogeneous) material has projected **thickness** $t(x,y)$ along the ray path and since δ and β are independent of the position we can write the **transmitted wave field** as

$$\Phi(x,y,z_0) = \exp\left[-k(i\delta + \beta)t(x,y)\right]\Phi(x,y,0),$$

The **intensity transmitted** is

$$I(x,y,z_0) = \exp\left[-\mu t(x,y)\right]I(x,y,0),$$

where $\mu = 2k\beta = 4\pi\beta/\lambda$ is the **attenuation coefficient**.

This equation above is simply the square modulus of the transmitted **wave field**; this equation is aka **Beer-Lambert law of attenuation.**

Deriving the transport of intensity equation

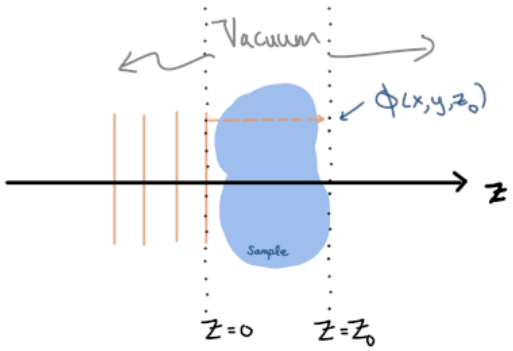
It is not possible to measure phase-contrast from a conventional radiograph of the sample. Any conventional detector will produce an intensity measurement that is proportional to the intensity of the complex wave field, therefore the information about the phase shift produced by the sample is lost. (Pelliccia et al. 976)

Assuming that the **projection approximation** holds and that the sample is in a vacuum so that the wave field propagation is only through the vacuum and the sample. This means that we can use the **paraxial homogeneous Helmholtz equation**

$$\left[\nabla_T^2 + 2ik\frac{\partial}{\partial z}\right]\Phi(x,y,z)e^{ikz} = 0$$

at any point after $z > z_0$

I want to attempt to derive the Transport of intensity equation as (Pelliccia et al. 976) describes



(49.19) envelope: $\Phi(x,y,z_0) = \sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}$ (intensity, phase)

(49.8) PHE : $\nabla_T^2\Phi + 2ik\frac{\partial\Phi}{\partial z} = 0$ (for vacuum)

using (49.19) in (49.8):

$$\nabla_T^2\Phi(x,y,z_0) + 2ik\frac{\partial\Phi(x,y,z_0)}{\partial z} = 0$$

$$\therefore \nabla_T^2\left(\sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}\right) + 2ik\frac{\partial\sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}}{\partial z} = 0$$

The issue is that I am getting a bit confused, because I am using the **complex envelope** as it is written in equation (49.19) in the text into the **homogeneous paraxial Helmholtz equation** as the text describes.

$$\therefore \underbrace{\nabla_T^2\left(\sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}\right)}_{①} + 2ik\underbrace{\frac{\partial\sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}}{\partial z}}_{⑥} = 0$$

$$① = \nabla_T\left(\underbrace{\nabla_T\left(\sqrt{I(x,y,z_0)}\right)}_{②} e^{i\varphi(x,y,z_0)} + \underbrace{\nabla_T\left(e^{i\varphi(x,y,z_0)}\right)}_{③}\sqrt{I(x,y,z_0)}\right)$$

$$② = \frac{1}{2}\left(I(x,y,z_0)\right)^{-\frac{1}{2}}\cdot\nabla_T\left(I(x,y,z_0)\right), \quad ③ = \underbrace{\frac{\partial}{\partial x}e^{i\varphi(x,y,z_0)}}_{④} + \underbrace{\frac{\partial}{\partial y}e^{i\varphi(x,y,z_0)}}_{⑤}$$

$$④ = i e^{i\varphi(x,y,z_0)}\cdot\frac{\partial\varphi(x,y,z_0)}{\partial x}, \quad ⑤ = i e^{i\varphi(x,y,z_0)}\cdot\frac{\partial\varphi(x,y,z_0)}{\partial y}$$

$$⑥ = \frac{\partial\sqrt{I(x,y,z_0)}\, e^{i\varphi(x,y,z_0)}}{\partial z} = 0$$

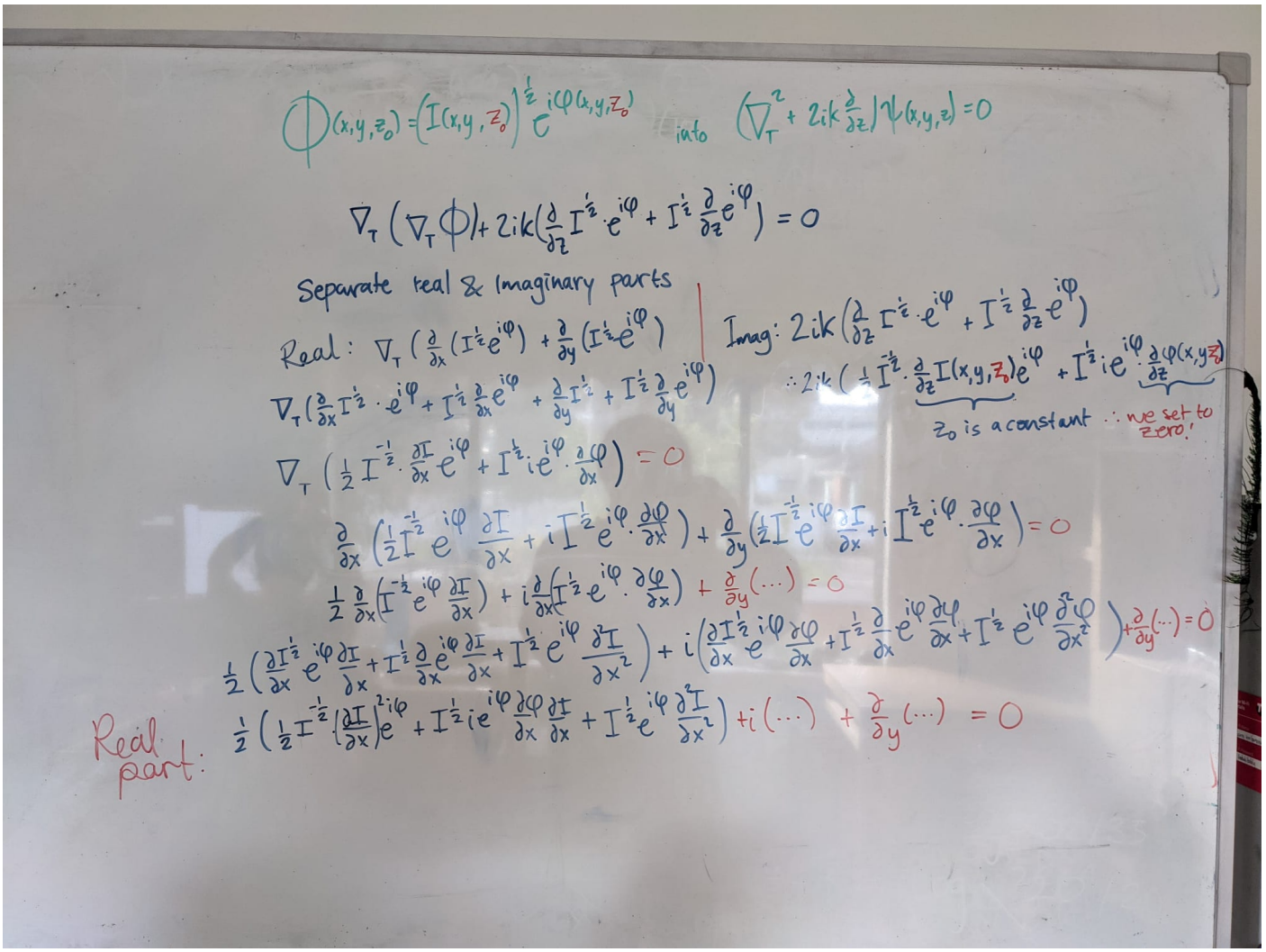
$$\nabla_T\left(\frac{1}{2}\left(I(x,y,z_0)\right)^{-\frac{1}{2}}\cdot\nabla_T\left(I(x,y,z_0)\right)e^{i\varphi(x,y,z_0)} + i e^{i\varphi(x,y,z_0)}\cdot\frac{\partial\varphi(x,y,z_0)}{\partial x} + i e^{i\varphi(x,y,z_0)}\cdot\frac{\partial\varphi(x,y,z_0)}{\partial y}\right) = 0$$

$$\therefore \nabla_T\left(\frac{i}{2}\left(I(x,y,z_0)\right)^{-\frac{1}{2}}\cdot\nabla_T\left(I(x,y,z_0)\right) + \frac{\partial\varphi(x,y,z_0)}{\partial x} + \frac{\partial\varphi(x,y,z_0)}{\partial y}\right) = 0$$

$$\therefore \nabla_T\left(\frac{i}{2}\left(I(x,y,z_0)\right)^{-\frac{1}{2}}\cdot\nabla_T\left(I(x,y,z_0)\right) + \nabla_T\varphi(x,y,z_0)\right)$$

powers of $\frac{1}{2}$, $-\frac{1}{2}$?

I just don't see how this is going to work out yet.



… I'll just **believe** the text for now.

21/07/21

Transport of intensity equation (**TIE**)

(I include here some **direct quotes** and **paraphrasing** from Transport of intensity equation: a tutorial. Optics and Lasers in Engineering 135 (**2020**) 106187 & Handbook of X-Ray Imaging: Physics and Technology. 1 ed., vol. 47, (CH.49))

The **TIE** is a second-order elliptic partial differential equation that describes the relationship between the **phase** $\varphi(x, y, z_0)$ and the axial **intensity** $I(x, y, z_0)$ variation of the optical field (Zuo et al. 9)

We write the **TIE** as

$$\nabla_T[I(x, y, z_0)\nabla_T\varphi(x, y, z_0)] + k\frac{\partial I(x, y, z_0)}{\partial z} = 0.$$

We see then a way to measure the phase shift imparted by the sample to an incident wave, by measuring the intensity of the transmitted and propagated wave field...
Clarification: The intensity measured after letting the beam propagate further downstream in empty space does contain phase information in it. But this information is generally ***encoded***, thus, it cannot be directly recovered during imaging (Pelliccia et al. 976).

Phase Retrieval

It turns out that measuring X-ray phase by means of the TIE is not quite straightforward: the phase cannot be measured directly relying on the TIE, only phase derivatives are accessible. * Let us take another look at Equation 49.20, this time let us expand the derivatives on the first term (Pelliccia et al. 976):

$$\nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi + k \frac{\partial I}{\partial z} = 0$$

where $k \frac{\partial I}{\partial z} << \nabla_T [I \nabla_T \varphi]$

i.e. the longitudinal variation of the envelope is smaller than the transverse variation at any $z > z_0$

$$\frac{\partial I}{\partial z} \approx \frac{I(x,y,z) - I(x,y,z_0)}{z - z_0}$$

$$\therefore \nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi + k \frac{\partial I}{\partial z} = 0$$

$$\therefore \nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi + k \left(\frac{I(x,y,z) - I(x,y,z_0)}{z - z_0} \right) = 0$$

$$\therefore \underbrace{I(x,y,z)}_{\text{measurable intensity for any } z > z_0} = \underbrace{I(x,y,z_0)}_{\text{Sample Attenuation (X-ray weak interaction with sample)}} - \underbrace{\frac{z - z_0}{k} (\nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi)}_{\text{phase derivatives terms}} \quad (99.23)$$

22/07/21

Propagation-based phase-contrast imaging aka (**PBI**)

$$\therefore \underbrace{I(x,y,z)}_{\text{measurable intensity for any } z > z_0} = \underbrace{I(x,y,z_0)}_{\text{Sample Attenuation (X-ray weak interaction with sample)}} - \underbrace{\frac{z - z_0}{k} (\nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi)}_{\text{phase derivatives terms}} \quad (99.23)$$

$$\underbrace{-\frac{z - z_0}{k} I \cdot \nabla_T^2 \varphi}_{\text{This term grows as distance from sample increases, if } z > z_0 \text{ is not too large}}$$

← modulates the attenuation image*

* responsible for edge-enhancement effect i.e. Fresnel diffraction in near-field regime

The edge-enhancement is generated because, in most cases, the transverse phase profile of a sample $\varphi(x, y, z_0)$ is smooth and slowly varying and, therefore, its Laplacian is negligible everywhere, except at the sample edges, where the phase profile changes abruptly (Pelliccia et al. 977).
The propagation distance enables the appearance of the sharp fringe at the boundaries between features of different refractive index (Pelliccia et al. 977).

Differential phase-contrast aka refraction phase-contrast (**RPC**)

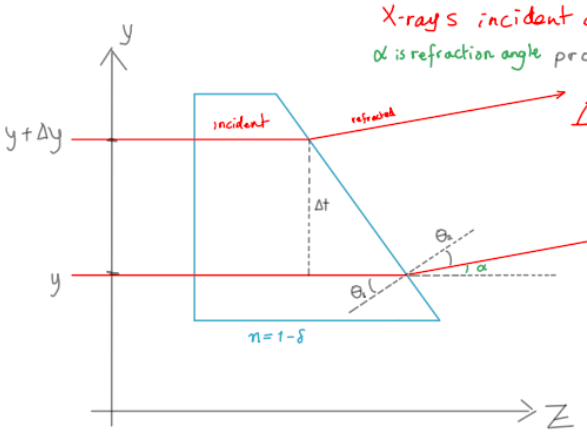
Generally requires X-ray optical elements of sorts, or a focused X-ray beam, to be visualized (Pelliccia et al. 977).

$$\therefore I(x,y,z) = I(x,y,z_0) - \frac{z-z_0}{k}(\nabla_T I \cdot \nabla_T \varphi + I \cdot \nabla_T^2 \varphi) \quad (99.23)$$

intensity — Attenuation

When a sufficiently strong intensity gradient is generated before or after the sample then the phase gradient can be recovered.

$$-\frac{z-z_0}{k}(\nabla_T I \cdot \nabla_T \varphi)$$



X-rays incident on a uniform non absorbing material

α is refraction angle proportional to $\nabla_T \varphi$

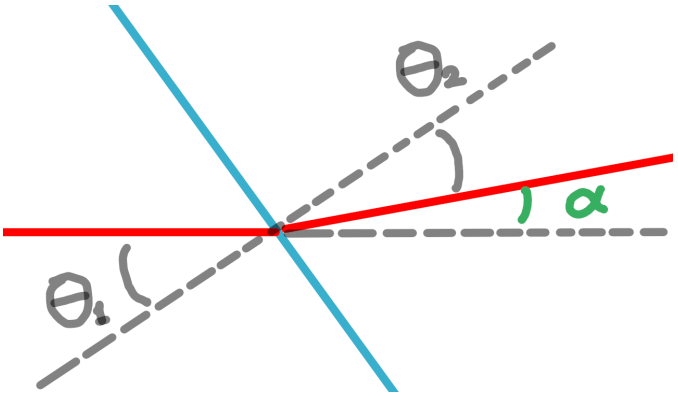
$\Delta\phi = \delta k \Delta t$ phase difference between wavefronts

$\delta = 1 - n$, $k = 2\pi/\lambda$, $\Delta t \approx \alpha \Delta y / \delta$ (for small displacements)

for HARD X-rays:

$\alpha = \theta_1 - \theta_2 << 1$

WHAT are these angles?



Snell's Law revision :P

$$(1-\delta)\sin\theta_1 = \sin\theta_2$$

$(1-\delta) = n_1$: Is the refractive index of sample

$1 = n_2$: Is the refractive index of the vacuum

θ_1: Is the angle of incidence wrt the **normal line** to the interface between the sample and the vacuum.

θ_2: Is the angle of refraction wrt the **normal line** to the interface between the sample and the vacuum.

α: Is the angle of refraction wrt the incident ray

$$\sin\theta_1 - \delta\sin\theta_1 = \sin\theta_2$$

$$\therefore \delta\sin\theta_1 = \sin\theta_1 - \sin\theta_2$$

Difference to product. $\sin\theta_1 - \sin\theta_2 = 2\cos\left[\frac{\theta_1+\theta_2}{2}\right]\sin\left[\frac{\theta_1-\theta_2}{2}\right]$

$$\therefore \delta\sin\theta_1 = 2\cos\left[\frac{\theta_1+\theta_2}{2}\right]\sin\left[\frac{\theta_1-\theta_2}{2}\right]$$

$$\therefore \delta\frac{\sin\theta_1}{\cos\left[\frac{\theta_1+\theta_2}{2}\right]} = \sin\left[\frac{\theta_1-\theta_2}{2}\right] \quad ?$$

I don't understand the role of the difference in thickness between the two positions of the incident rays Δt.

All differential phase-contrast techniques are realized by introducing an angular sensitivity in the imaging set-up (Pelliccia et al. 977).

The measurement of the refraction angle in any given direction yields the differential phase in the same direction (Pelliccia et al. 978).

It is common to observe both propagation phase contrast (**PB**) and refraction phase-contrast (**RPC**)

I get it now! Since the **small angle approximation** is a basic assumption of the **paraxial approximation** then I can use it in my derivations:

$$(1-\delta)\sin\theta_1 = \sin\theta_2 \quad (49.24)$$

$$\alpha = \theta_1 - \theta_2 << 1$$

$$\sin\theta_1 - \delta\sin\theta_1 = \sin\theta_2$$

$$\therefore \delta\sin\theta_1 = \sin\theta_1 - \sin\theta_2$$

Small angle approximation: $\tan\theta_1 \approx \sin\theta_1 \approx \theta_1$

$$\therefore \delta\tan\theta_1 \approx \theta_1 - \theta_2$$

$$\therefore \delta\tan\theta_1 \approx \alpha \quad (49.25)$$

for small displacements the difference in thickness: $\Delta t \approx \alpha\Delta y/\delta$

$$\therefore \Delta t \approx \tan\theta_1 \Delta y$$

The phase difference: $\Delta\phi = \delta K \Delta t$

$$\therefore \Delta\phi = \delta K \tan\theta_1 \Delta y$$

$$\therefore \frac{\Delta\phi}{\Delta y} = K\alpha \quad (49.26)$$

The measurement of the refraction angle in any given direction yields the differential phase in the same direction (Pelliccia et al. 978).

...since both propagation based phase-contrast and refraction contrast require a non-zero propagation distance, z (see Equation 49.23), it is normal to observe both effects concurrently during the same measurement (Pelliccia et al. 978).

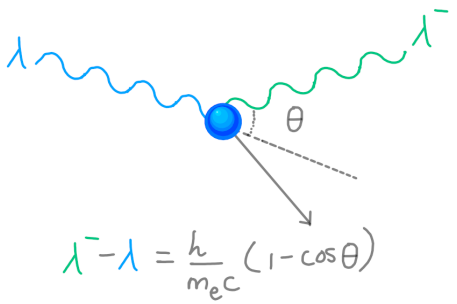
Introduction to visualising X-ray Scattering

(I include here some **direct quotes** and **paraphrasing** from Handbook of X-Ray Imaging: Physics and Technology. 1 ed., vol. 47, (CH.49) and Wikipedia)

X-rays scatter in both elastic and inelastic processes.

$< 30keV$	$> 30keV$
elastic scattering is predominant	**inelastic** scattering is predominant

In **Compton scattering** X-rays undergo **incoherent** scattering in all directions regardless of the direction of the incident wave



where
λ is the initial wavelength,
λ' is the wavelength after scattering,
h is the Planck constant,
m_e is the electron rest mass,
c is the speed of light, and
θ is the scattering angle.

(Wikipedia, the free encyclopedia)

Compton scattering is a nuisance in medical imaging, since the halo associated with the **incoherent** scattering degrades the imaging contrast (Pelliccia et al. 978)

Elastic processes are “coherent”, thus, the direction of the scattered X-rays is related to the incident direction and the sample morphology (Pelliccia et al. 978)

PB X-ray imaging

Phase-contrast can be considered aberration to an imaging system (not truly representing the object). It is often considered important to quantitatively recover information about the object from the obtained images.

The exit-surface wavefield provides a **true image** of the object, free of phase-contrast aberrations, and can reveal quantitative information about the object's composition. However, it is currently impossible to directly recover this information, as X-ray detectors have only ever been capable of measuring the intensity (amplitude squared) and not the phase Information. Recovery of this lost phase information from intensity data alone is an ill-posed problem(?), and is commonly known as **phase retrieval** (Pelliccia et al. 978).

Different phase-retrieval algorithms for **PB** exist.
Here are some of the **characteristic limitations** of these algorithms:

- Limiting the samples to those that minimally interact with X-rays.
- Requiring that the sample is composed of a single **monomorphous** (homogeneous) material

Important questions

- Am I missing a factor of k in the **PiHE** in the second term?

$$2ik\frac{\partial\Phi(x,y,z)}{\partial z} + (n^2(x,y,z)-1)\Phi(x,y,z) = 0$$

Week 1 (26/07/21 –01/08/21)

Aims:

1. Understand the mathematical tools of
    a. **X-ray phase-contrast** effects in imaging
    b. methods to extract quantitative information from phase-contrast data.
2. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Getting familiar with the **Transport-of-Intensity equation**
2. ~~Formalise weekly meeting schedule. Our first meeting will occur on **27th or 28th july (TBC)**~~
3. **Read X-Ray Handbook**
4. Get through slides/lectures for each of the eight topics from **PHS4200/PHS4020**

26/07/2021

Lecture Notes 1

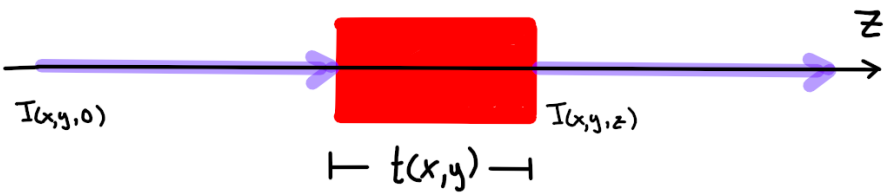
(I include here some direct **quotes, paraphrasing**, image **screenshots,** and refactored **drawings** from the lectures on X-ray optics from MK's & Kaye Morgan (KM) honours course PHS4200. *)*

In the first Lecture "Optics Fundamentals". MK's recommended texts include David' Paganin's (DP) Coherent X-ray optics (2006), which I will search for on the uni library website.

MK mentioned:

- **Beer-Lambert Law** (X-ray attenuation)
  (exponential decay through a material thickness)

$$I(x,y,z_0) = \exp\left[-\mu t(x,y)\right] I(x,y,0),$$



The attenuation coefficient (μ) changes as the material and its density changes i.e.

$$\mu \sim\propto \frac{Z^3}{E^3}$$

Where Z is the atomic number of the material and E is the energy (this wasn't entirely clear to me… the energy of x-rays?)

- Sweet Balance (X-ray interactions with matter)
  Low E -> **Low attenuation and High contrast** -> **High dose absorption** **(radiation risk)**
  High E -> **Long scale of contrast** -> **Low image contrast**

- Contrast is the difference in intensity.
- Intensity varies due to
  - Absorption
  - Elastic scattering
  - Inelastic (compton scattering)

IF random photons hitting detector in random positions due to scattering
THEN noise in image degrading quality
solution?

- **Synchrotron** - Collimation of electrons in circular orbit due to lorentz contraction

Information on the shape of objects can be lost on the 1D image obtained with X-rays.
We have strong dependence on material density and/or thickness only. Not necessarily on the arrangement of the object and its shape.

Image Quality

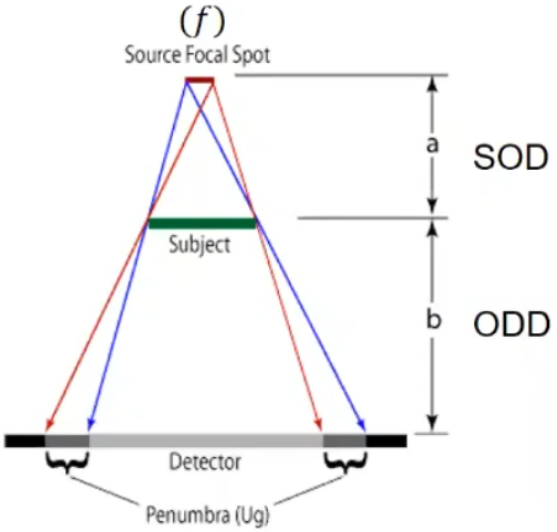
is generally defined in terms of

1. **NOISE** - visible photon pixelation effect on image (stochastic process)
   a. Several images can be taken and averaged out to fix noise
   b. Signal-to-Noise ratio (**SNR**)
      i. Poisson statistics -> noise is $\sqrt{N}$ for N photons
      ii. **SNR ->** $N/\sqrt{N} = \sqrt{N}$
2. **SPATIAL RESOLUTION** - **How many pixels are displayed per unit distance of an image**.
3. **CONTRAST RESOLUTION** - difference in image density between adjacent areas of an image
   a. **Subject contrast** - physical variation of density within the sample/patient
   b. Detector contrast -contrast capabilities of apparatus detecting photons
   c. Display contrast - reliability/representativeness of the image as it is displayed.

Factors affecting image formation quality

- Geometric factors - Magnification
  - **SID**: source-to-image distance
  - **SOD**: source-to-object distance
  - **MF**: magnification factor
    - $\mathrm{MF} = \mathrm{SID}/\mathrm{SOD}$

- Geometric “unsharpness” - depends on magnification



Penumbra region is an undesirable consequence of large **SID**. A way to decrease unsharpness (penumbra) is to decrease **ODD**, hence decreasing magnification of the image.

27/07/2021

PB X-ray imaging (continued)

Different phase-retrieval algorithms for **PB** exist.
Here are some of the **characteristic limitations** of these algorithms:

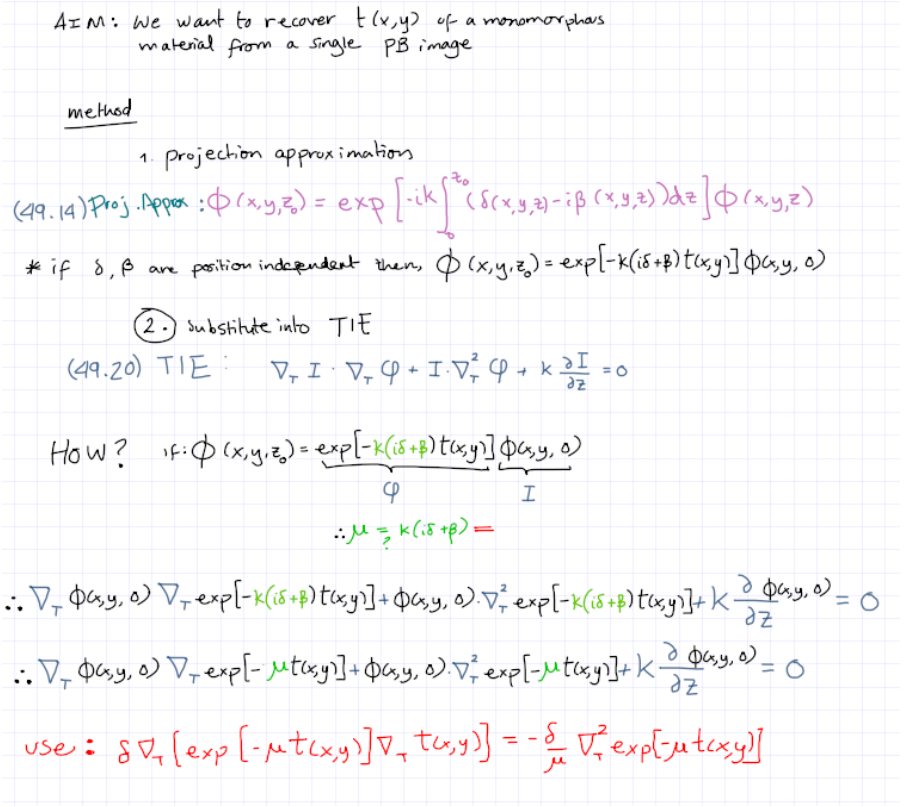
- Limiting the samples to those that minimally interact with X-rays.
- Requiring that the sample is composed of a single **monomorphous** (homogeneous) material

Using the projection approximation (Paganin et al. 2002) showed that it is possible to accurately recover the projected thickness of a monomorphous object from just a single propagation-based phase-contrast (PB) image (Pelliccia et al. 979).

Using Paganin's algorithm requires *a priori* knowledge of

1. μ (attenuation coefficient)
2. δ (refractive index decrement)

Also radiation **bandwidth** should be minimal (small range of allowed frequencies in the radiative signal?)
I am following the notes from the Handbook in order to understand the underlying mathematics from Paganin's algorithm.

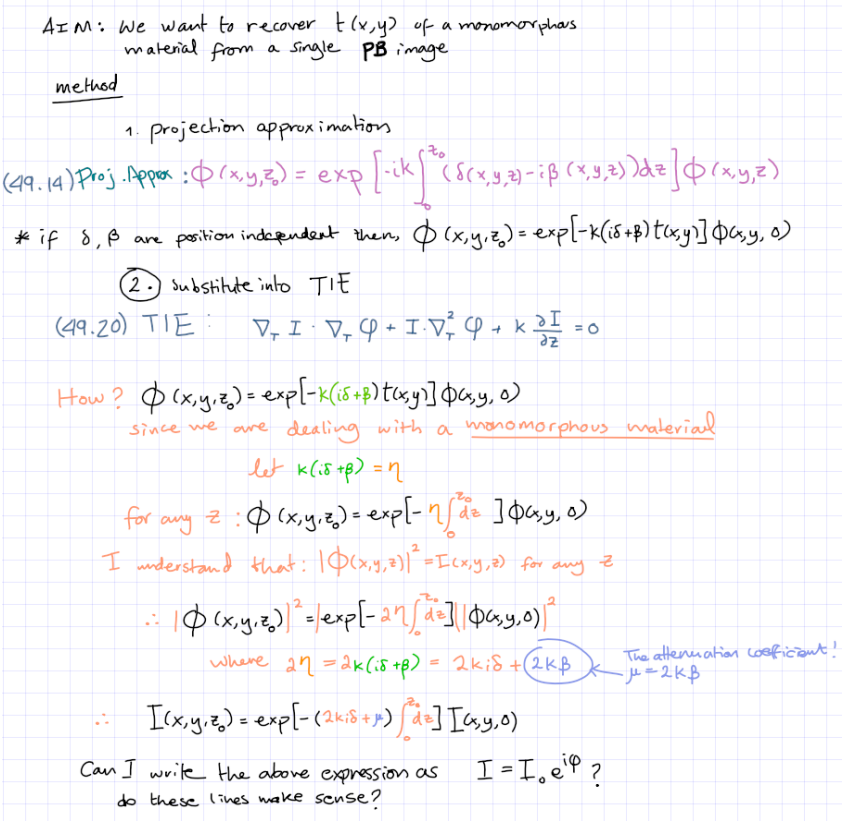


I've been thinking about this for a few hours now… I guess I am stuck.

28/07/21

MEETING DAY!

I am putting all the thoughts into writing, but I am still unsure of the process I must follow.



Maybe I'll ask MK if my ideas make sense because so far it feels a bit convoluted.

Questions for MK:

1. Is the **projection approximation** missing a factor of k squared in the second term?
    a. If not… how did that factor get deleted?
    This was a typo in the Handbook. The correct expression is
$$2i\frac{\partial\Phi(x,y,z)}{\partial z}+k(n^2(x,y,z)-1)\Phi(x,y,z)=0$$
2. How does one derive the **TIE?**
    *MK pointed to topic 4 in PS4200 Lectures for a derivation of the **TIE***
3. Paganin's algorithm.
    i. What happens with the δ(refractive index decrement) term?
    Mod squared makes imaginary part in phase factor disappear
    ii. Projection approximation substitution into TIE. How?
    iii. How do I use identity (49.27)?
4. What else should I do?
    a. The aim of my first mini project is to investigate via a simulation how phase changes as density changes in a material. This investigation can be extended by also simulating two distinct materials and see if phase contrast occurs. Later I can test if we can do successful phase retrieval in this scenario.

    In essence I want to verify this claim in the Handbook:
    ((On Paganin's algorithm)) also depends on the
    ratio between δ and μ, both of which are proportional to the density
    of the medium, hence changes in material density throughout
    the material are permitted (Pelliccia et al. 979).

    I want to see if that ratio is actually a ratio of differences i.e. $\frac{\Delta\delta}{\Delta\mu}$.

    How much density difference is needed before I see phase contrast

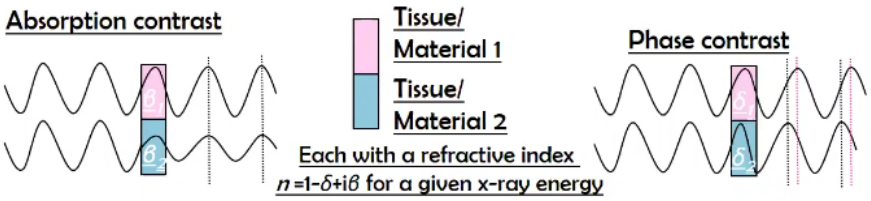
    b. Get into Beltran et al. 's 2010 paper to check Paganin's phase retrieval algorithm extension, in order to reconstruct boundaries?
    c. EXPERIMENTAL: Help MK with his test on the silver target and data analysis to compare the results from the silver target to the classic tungsten target.
    d. MORE SIMULATIONS: Speckle analysis. What causes speckles in images?

One-on-one meeting schedule: Wednesdays 11am
Group meeting schedule: Wednesdays 1pm
I must start planning out my simulation.

29/07/2021

Lecture Notes 2

(I include here some direct **quotes, paraphrasing**, image **screenshots,** and refactored **drawings** from the lectures on X-ray optics from MK's & Kaye Morgan (KM) honours course PHS4200. *)*



Some beams are attenuated more than others. Also phase can be shifted in some rays.

Higher X-ray energy -> smaller refraction angles

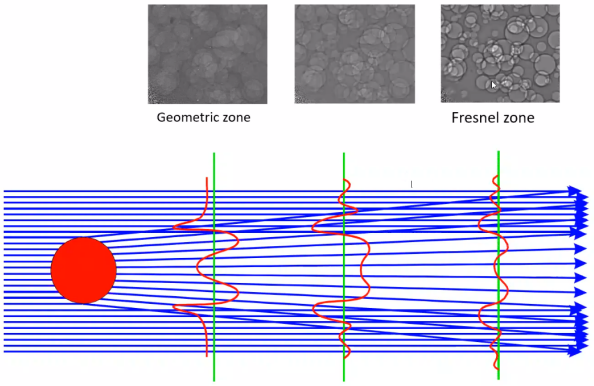
Refractive index: $n(x,y,z)=1-\delta+i\beta$

Where δ is related to the **refraction** (**dispersion**), and β to the **absorption**, $\delta,\beta<<1$

PBI

If we have our detector close to the sample we don't get a lot of contrast effects.
As we move our detector away we get better contrast with the edges… and more diffraction.

In the **Geometric zone** in the figure below we do not have as much information about the pattern (in this region most of the contrast is due to absorption) as in the **Fresnel zone** where phase contrast reveals the structure of the sample.



Differential cross section (Scattering)

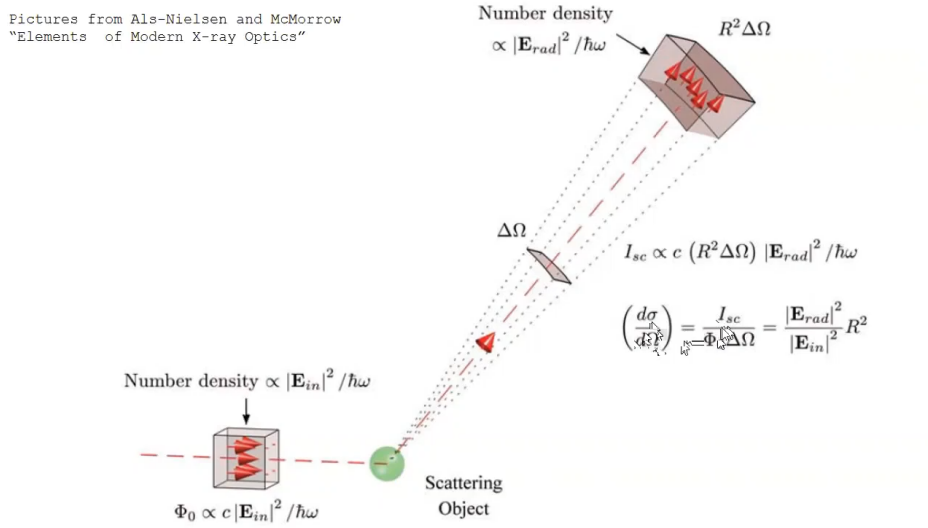
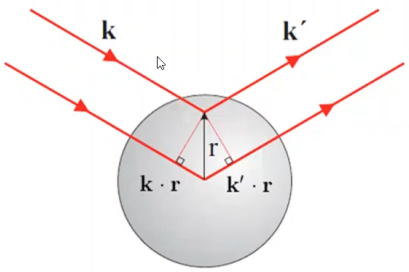


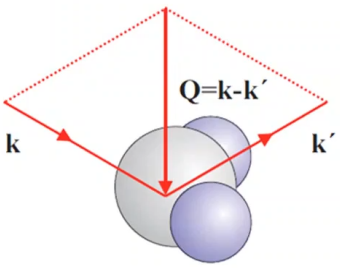
Fig. 1.5 Schematic layout of a generic scattering experiment used to determine the differential cross-section ($d\sigma/d\Omega$): see Eq. (1.2). The incident beam flux Φ_0 is the number of particles per second per unit area. For an electromagnetic wave this is proportional to $|\mathbf{E}_{in}|^2$ times the velocity of light, c. The incident beam interacts with the target object to produce the scattered beam. A detector records the scattered intensity, I_{sc}, defined as the number of counts recorded per second, which is proportional to $|\mathbf{E}_{rad}|^2$ times the area of the detector and the velocity of light. The detector is located a distance R from the target object, and subtends a solid angle of $\Delta\Omega$.

The cross sectional area is a measure of the efficiency of the scattering process. Specifically of the incident flux from the beam.

Scattering by free electrons is aka **Thompson Scattering**

ATOMIC FORM FACTOR





The term $f^0(\mathbf{Q})$ is called the "atomic form factor"

$$f^{(0)}(\boldsymbol{Q}) = \int \rho(\boldsymbol{r}) e^{i\boldsymbol{Q}\cdot\boldsymbol{r}} d\boldsymbol{r}$$

This formula represent the Fourier transform of the electron density.

In the limit that $\boldsymbol{Q} \to 0$ all of the different volume elements scatter in phase so that $f^0(\boldsymbol{Q}=0) = Z$, the number of electrons in the atom.

As $\boldsymbol{Q}$ increases from zero the different volume elements start to scatter out of phase and consequently $f^0(\boldsymbol{Q} \to \infty) = 0$.

Forward scattering tells us about the electron density

The Complex Refractive Index

If the refractive index n is now allowed to be a complex number,

$$n = 1 - \delta + i\beta,$$

then the wave propagating in the medium is:

$$e^{inkz} = e^{i(1-\delta)kz} e^{-\beta kz}$$

Consider the intensity $I = \left|e^{inkz}\right|^2 = e^{-2\beta kz} = e^{-\mu z}$

From this equation for the amplitude it can be inferred that $\beta = \mu/2k.$

And we already know that $\delta = \dfrac{2\pi\rho r_o}{k^2}$

The Complex Refractive Index – Molecules

For molecules we have to sum over multiple atoms of j:

$$\delta = \left(\frac{2\pi r_0}{k^2}\right)\sum_j \rho_{at\_j}\{f^0(0) + f'\}_j$$

In the forward direction, far from resonance:

$$\delta = \left(\frac{2\pi r_0}{k^2}\right)\sum_j \rho_{at_j} Z_j = \frac{2\pi r_0 \rho}{k^2} \quad \text{where } \rho = \text{electron density}$$

$$\mu = -\left(\frac{4\pi r_0}{k}\right)\sum_j \rho_{at_j} f_j'' = \sum_j \rho_{at_j}\sigma_{a_j} \approx \underbrace{\frac{K_1\rho}{A} f_{KN}(E)}_{\text{Compton Scattering}} + \underbrace{\frac{K_2}{AE^3}\sum_j \rho_{at_j} Z_{at_j}^n}_{\text{Photoelectric Absorption}};\ n{\sim}3$$

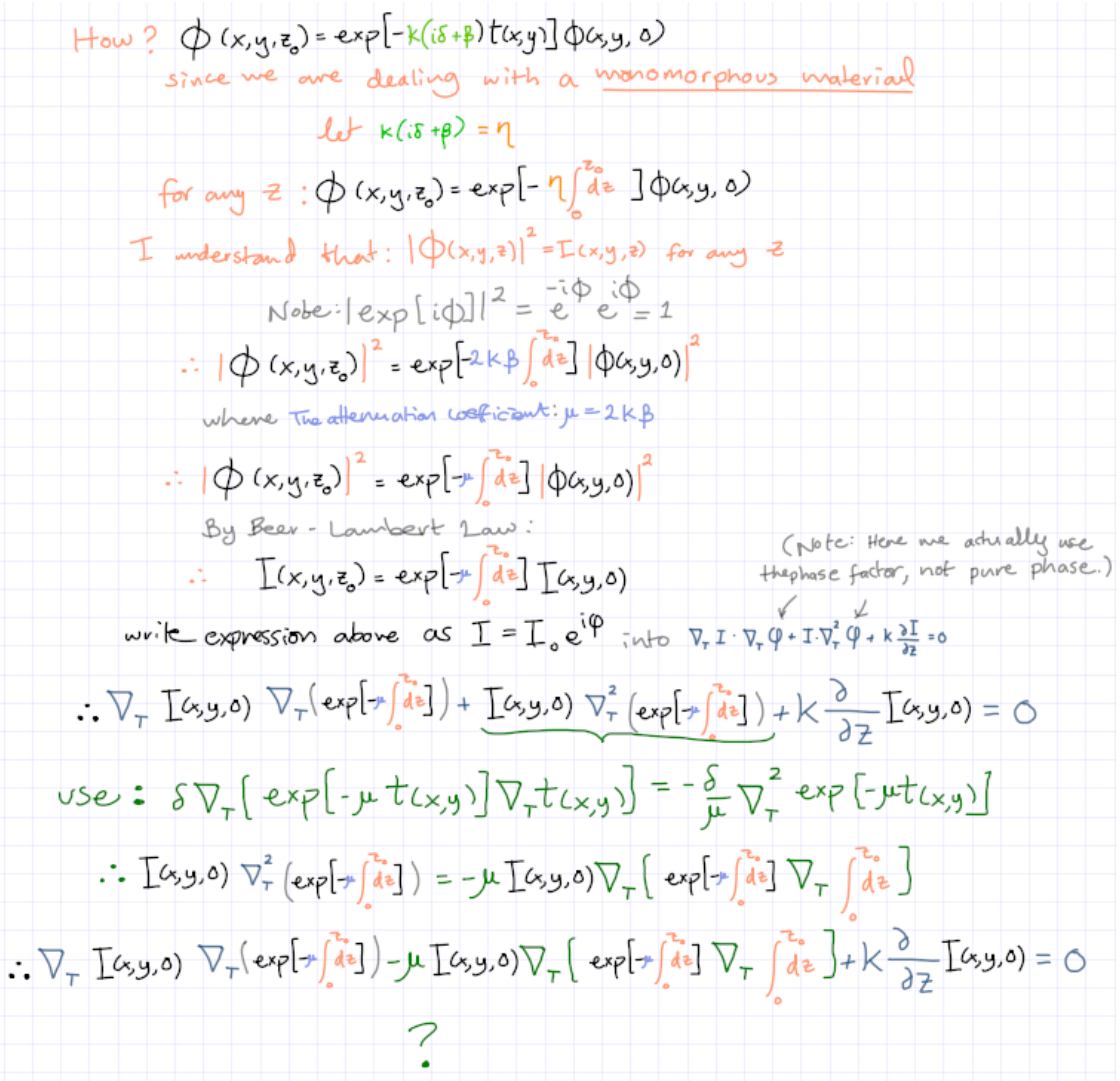
See also: Alvarez and Makovski, Phys. Med. Biol., **21**, 733, 1976.

For coefficients see the NIST database: https://www.nist.gov/pml/x-ray-and-gamma-ray-data

Note: The above equations assume individual atoms scatter independently, with neighbouring atoms having no effect; a valid approximation at energies above 50 eV and far from absorption edges.

31/07/2021

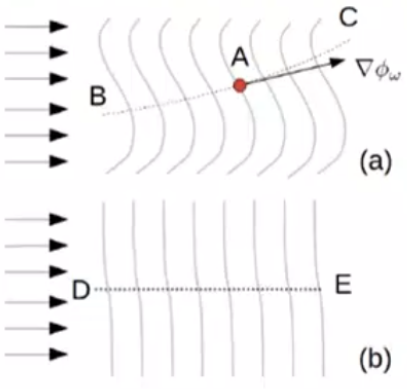
Here is another attempt at recovering the projected thickness of a monomorphous object from just a single propagation-based phase-contrast (PB) image: I think It is making more sense but I'm still confused.



Lecture Notes 3

(I include here some direct **quotes, paraphrasing**, image **screenshots,** and refactored **drawings** from the lectures on X-ray optics from MK's & Kaye Morgan (KM) honours course PHS4200. *)*

The Poynting Vector



The Poynting vector shows the direction of energy flow. The gradient operator produces a vector parallel to the normal of the surface being differentiated, in this case the surface of constant phase.

$$\boldsymbol{S}(x,y,z,t) = \frac{1}{\mu_0}\boldsymbol{E}(x,y,z,t)\times\boldsymbol{B}(x,y,z,t)$$

$$\boldsymbol{S}(x,y,z,t) \propto I(x,y,z,t)\boldsymbol{\nabla\phi}(x,y,z,t)$$

FIG. 1 X-ray wave-fronts. (a) At a given instant of time $t = t_0$, the intensity of an X-ray field is $I(x,y,z,t = t_0)$. The wave-fronts, namely the surfaces of constant phase $\phi(x,y,z,t = t_0)$, are indicated by the series of curved surfaces. At any point A the wave-fronts move away from the source as time increases. The direction of energy flow at point A is $\nabla\phi(x,y,z,t = t_0)$. The associated current density (Poynting vector) is $\mathbf{S}(x,y,z,t = t_0) \propto I(x,y,z,t = t_0)\nabla\phi(x,y,z,t = t_0)$, which is perpendicular to each wave-front. The current density is everywhere tangent to the associated streamlines, such as the streamline BAC passing through the point A. (b) For a paraxial field, all wave-fronts may be obtained by slightly deforming planar surfaces perpendicular to the optic axis (*i.e.* the z axis, which runs from left to right). All Poynting vectors are close to being parallel to the optic axis (hence the term paraxial). The associated streamlines such as DE are well approximated by straight lines parallel to the optic axis. Image adapted from Paganin (2006).

Paganin and Pelliccia, p4. 10

This is why we want to use the **PiHE!**

01/08/2021

I chatted to Chris about the best way to do a simulation of X-ray propagation through a couple of concentric cylinders of different materials/densities, and he suggested that I learn about how to use PyQt... so today I am gonna read: **Mark Summerfield's (2015) - "Rapid gui programming with python and qt. The definitive guide to pyqt programming".**
I plan to do some basic GUIs and see if I get a basis to start up my simulation.
Today I also managed to access the Confluence -"X-ray imaging group" page, and I am going over the info they have on python to give me ideas of the possible visualisation tools that the group already uses.
...After a quick browse… nothing in the imaging group's confluence page appears relevant to my current state.

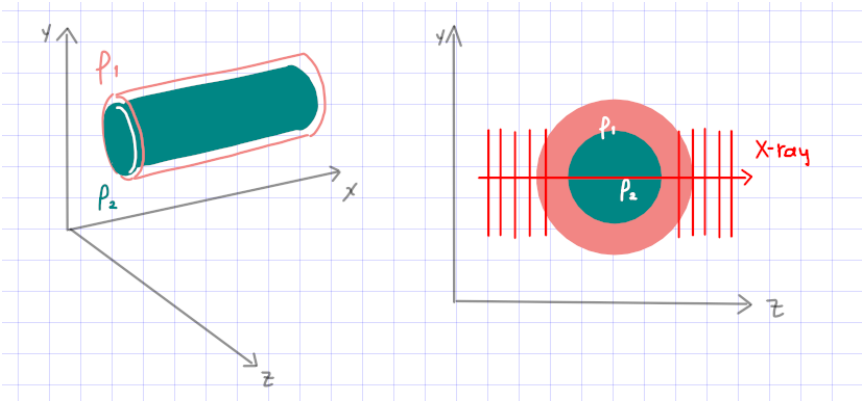
Simulation design ideas

e. The aim of my first mini project is to investigate via a simulation how phase changes as density changes in a material. This investigation can be extended by also simulating two distinct materials and see if phase contrast occurs. Later I can test if we can do successful phase retrieval in this scenario.

In essence I want to verify this claim in the Handbook:
((On Paganin's algorithm)) also depends on the
ratio between δ and μ, both of which are proportional to the density
of the medium, hence changes in material density throughout
the material are permitted (Pelliccia et al. 979).

I want to see if that ratio is actually a ratio of differences i.e. $\frac{\Delta\delta}{\Delta\mu}$.

How much density difference is needed before I see phase contrast



I want to use a GUI program so that the user can interactively modify the density of the concentric tubes in the simulation.

Week 2 (02/08/21 –08/08/21)

Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. ~~Meetings on **Wednesday 04/08**~~
   a. ~~one-on-one: **11 am**~~
   b. ~~Group: **1 pm**~~
2. ~~Getting familiar with the **Transport-of-Intensity equation** usage~~
3. **Read X-Ray Handbook**
4. **Read PyQt GUI book**
5. ~~**Read Beltran et al. (2010)**~~
6. ~~Get through slides/lectures for the first 4 topics from **PHS4200/PHS4020**~~
7. ~~Watch Pags Lectures on X-ray Phase contrast imaging (3, 4, 5, 6)~~

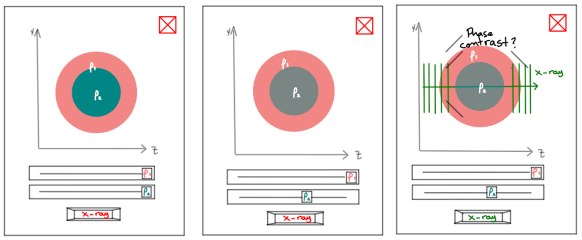
02/08/2021

Simulation design ideas (continued)

I've been going through some examples in the PyQt GUI book, and I have a draft script of what I want to create:

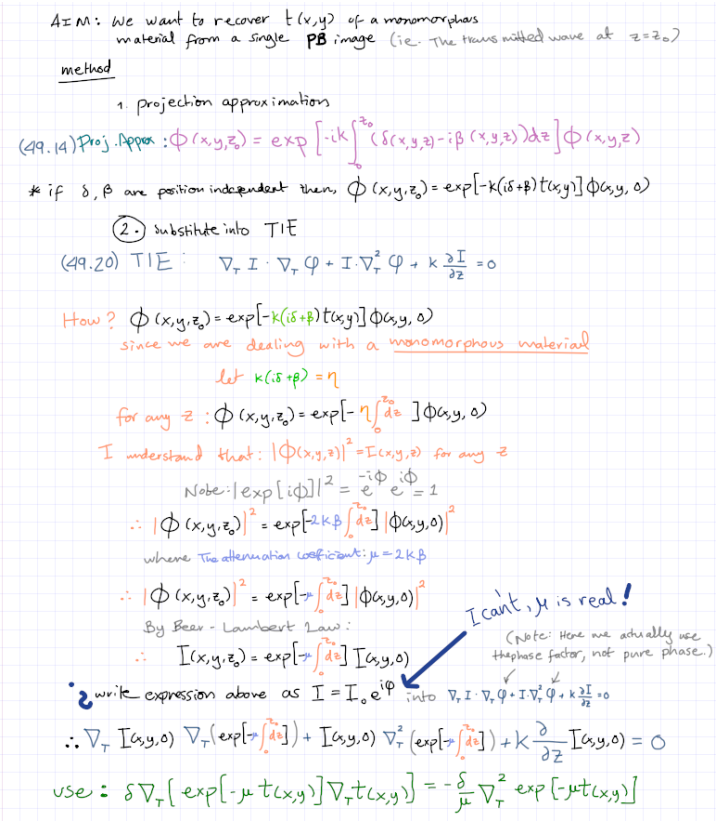
```
import sys
from math import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class Form(QDialog): # Blank form rectangle

    # def returnPressed(self): # consider what kind of signal I might need to update GUI with slider widgets
    #     self.label.setText(self.lineedit.text())

    def __init__(self, parent=None): # Method: default parent of None
        super(Form, self).__init__(parent) # TOP level window

        # self.browser = QTextBrowser() # make this a viewer widget instead
        # self.lineedit = QLineEdit("Type an expression and press Enter") # make this slider widgets instead
        # self.lineedit.selectAll() # ?

        layout = QVBoxLayout() # arrangement of the window box widgets
        # layout.addWidget(self.browser) # modify this for sim viewer and density sliders
        # layout.addWidget(self.lineedit)

        self.setLayout(layout) # defaults ownership of the widgets and itself to the Form
        self.lineedit.setFocus() # ?
        # self.lineedit.returnPressed.connect(self.updateUi) # SIGNAL connect to updateUi
        self.setWindowTitle("Phase contrast & material density") # window title

    def updateUi(self):
        try:
            # investigate slider usage on viewer
        except:


    def TIE():
        # insert maths for X-ray propagation
    def drawing_cylinders():
        # visual representation in viewer

app = QApplication(sys.argv)
form = Form()
form.show()
app.exec_()
```

This is more or less the simulation interface that I want to make:



I want to solve the **TIE** and recover the phase after the plane X-ray wavefronts have passed through the projected thickness of the cylinders (I want to do this everytime the user updates the density parameters of the cylinders in the GUI). Since the μ, δ parameters are proportional to the density of the cylinders. I must find a way to connect the user signal in the GUI to the **TIE** explicitly.

03/082021



I'll just proceed anyway.…
The Handbook claims that using identity (49.27) in the TIE yields equation (49.28):

$$-\frac{\delta}{\mu}\exp\left[-\mu t(x,y)\right] = \frac{\partial}{\partial z}I(x,y,0)$$

Where the RHS is approximately the intensity difference between the contact plane and that of the phase contrast divided by the distance between the two planes Δ (Pelliccia et al. 979).

Using the **Fourier Derivative theorem** $\mathfrak{F}\left\{f^{(m)}\right\} = (ik)^{(m)}\mathfrak{F}\{f\}$ on this DE yields the **projected thickness**:

$$t(x,y) = -\frac{1}{\mu}\log\left[\mathfrak{F}^{-1}\left[\frac{\mathfrak{F}\left[I(x,y,z=\Delta)\right]/I_0}{\Delta\delta\left|\mathbf{k}_T\right|^2/\mu+1}\right]\right],$$

where k_T is the transverse (dual) Fourier space coordinate.

So this is the point at which the Handbook mentions that Paganin's algorithm

also depends on the ratio between δ and μ, both of which are proportional to the density of the medium, hence changes in material density throughout the material are permitted (Pelliccia et al. 979).

I want to see if that ratio is actually a ratio of differences i.e. $\frac{\Delta\delta}{\Delta\mu}$.

I am a bit unsure at this stage of what I have to use as my mathematical tools.

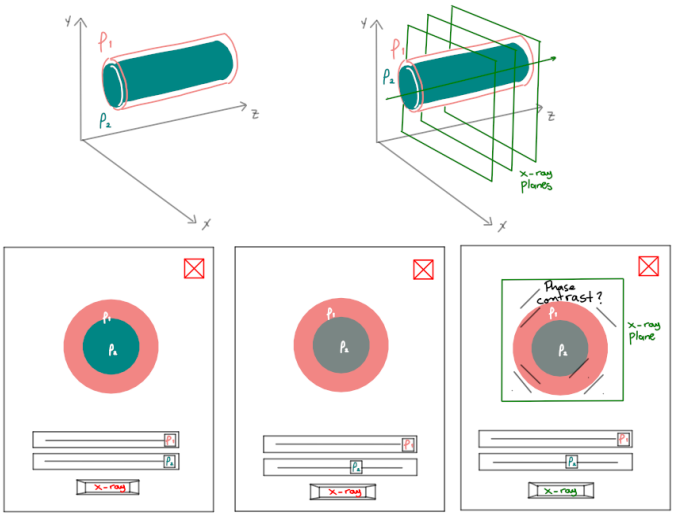
1. **TIE (space propagation)**

$$\nabla_T[I(x,y,z_0)\nabla_T\varphi(x,y,z_0)] + k\frac{\partial I(x,y,z_0)}{\partial z} = 0.$$

   To obtain the exit-surface phase information.
2. I need to check out the extended phase retrieval algorithm by **Beltran et al. (2010)**
   Their approach requires just a single phase-contrast projection, and enables the user to focus on a material of interest (a) embedded in an encasing medium (b). The complex refractive index of all materials needs to be known a priori, as does the total projected thickness of the sample, A(x, y) (Pelliccia et al. 980)

   Beltran's method requires a priori knowledge of:
   1. The complex refractive index for each material present
   2. The total projected thickness of the object at each orientation

Wo wo wo I just thought of something... I think I've made a mistake in my design sketches. I possibly confused my coordinates x and z. If so, this is the correct design:



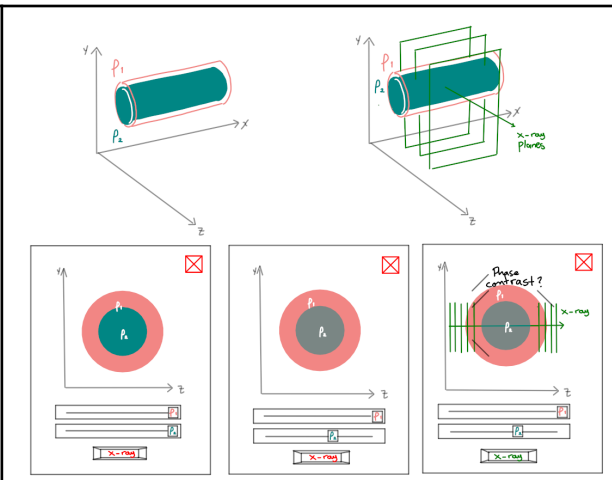
I am thinking that I am going to have to use some propagation scheme to sort that X-ray motion in z.
I should use the Fourier derivative theorem on the TIE just like Pags.

04/08/21

MEETING DAY!

Questions for MK:

1. Paganin's algorithm.
   i. Projection approximation substitution into TIE. How?
   ii. How do I use identity (49.27)?
2. Sketches of GUI design???

So… No, I didn't confuse my coordinates x and z. My design should look like this:	

Marcus suggested that I use the projection approximation (object in 2D)
NOTE: \Delta \phi = -k delta T where T is projected thickness

I need to figure out how to use the projection approximation so that the 2D array follows the projected thickness of a cylinder

To break the simulation into manageable bits I could create:

1. ***a single cylinder***
2. ***cylinder in box***
3. ***cylinder in cylinder***

Lecture Notes 4 COHERENT X-RAY IMAGING & TIE

(I include here some direct **quotes, paraphrasing**, image **screenshots,** and refactored **drawings** from the lectures on X-ray optics from MK's & Kaye Morgan (KM) honours course PHS4200. *)*

Previously we saw that inside an object, under the paraxial approximation and ignoring transverse energy flow, that we can determine the wavefield at the object's exit surface. We call this the Projection approximation:

$$\underbrace{\Phi(x,y,z=z_0)}_{\text{Output (field)}} \approx \underbrace{\exp\left\{-ik\int_0^{z_0}[\delta(x,y,z)-i\beta(x,y,z)]\partial z\right\}}_{\text{Operator (a.k.a. Transfer function) This is the optical element, in this case just the object of interest.}} \underbrace{\Phi(x,y,z=0)}_{\text{Input (field)}}.$$

$$\Delta\phi(x,y,z=z_0) \equiv \text{Arg(transfer or transmission fn)} = -k\int_0^{z_0}\delta(x,y,z)\partial z$$

$$I(x,y,z=z_0) = \Phi(x,y,z=z_0)\Phi^*(x,y,z=z_0) =$$

$$= I(x,y,z=0).\exp\left\{-2k\int_0^{z_0}\beta(x,y,z)\partial z\right\} = I(x,y,z=0).\exp\left\{-\int_0^{z_0}\mu(x,y,z)\partial z\right\}.$$

for a single material (?) $-k\delta f(x,y)$

$$\Delta\phi(x,y,z=z_0) \equiv \text{Arg(transfer or transmission fn)} = -k\int_0^{z_0}\delta(x,y,z)\partial z$$

Beer Lambert Law → $I(x,y,z=z_0) = \Phi(x,y,z=z_0)\Phi^*(x,y,z=z_0) =$

$$= I(x,y,z=0).\exp\left\{-2k\int_0^{z_0}\beta(x,y,z)\partial z\right\} = I(x,y,z=0).\exp\left\{-\int_0^{z_0}\mu(x,y,z)\partial z\right\}$$

← phase information ($\Delta\Phi$) is LOST!

By operating on $\underbrace{\Phi(x,y,\xi)}_{\text{Output (field)}} = \underbrace{\boldsymbol{O}(x,y,\xi)}_{\text{Operator - (Transfer Function)}}\underbrace{\Phi(x,y,z=z_0)}_{\text{Input (field)}}.$ we can entangle $\Delta\phi$ with

The exit surface Intensity ⇒ Recover $\Delta\phi$

$$I(x,y,\xi) = \Phi(x,y,\xi)\Phi^*(x,y,\xi) = [\boldsymbol{O}(x,y,\xi)\,\Phi(x,y,\xi)][\boldsymbol{O}^*(x,y,\xi)\Phi^*(x,y,\xi)]$$

One solution for calculating the intensity downstream of an object is the TIE (Teague, 1983). This is a continuity equation similar to the Schrodinger equation, based on a hydrodynamic formulation, but for E/M waves. It's all about conservation of energy flow.

$$\underbrace{-\nabla_\perp \cdot}_{A}\ \underbrace{[I(r)\nabla_\perp \phi(r)]}_{B} = \underbrace{\frac{k\partial I(r)}{\partial z}}_{C}$$

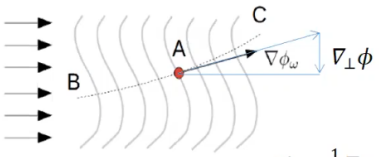
A. Convergence of B. Flow of energy = C. Increase in intensity (and vice verse).
That is, optical focussing/spreading of energy.

Note the negative sign.. If we have a decrease of intensity in the RHS we have a divergence in the LHS i.e. nabla is the divergence operator

How do I solve the **TIE at a specific point in the z direction?**

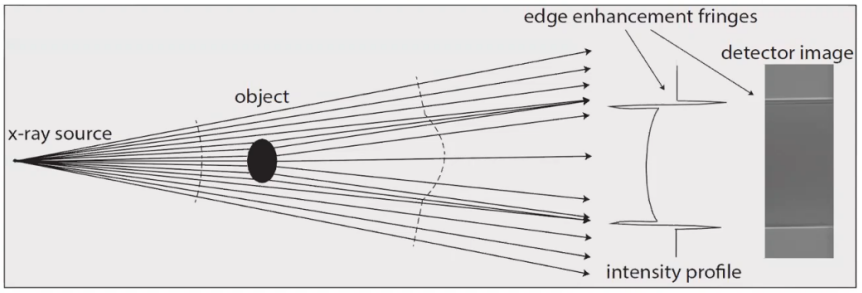
MK suggests finite differences… (Propagation based imaging - free space propagation operator)
Therefore I can do a higher order finite difference approach such as **Runge Kutta**!

Link to Geometrical Optics Approximation (GOA)



A plane wave $\exp(ikz)$ has $\phi = kz$. Thus $\nabla\phi = k\hat{z}$, hence it travels travelling in direction $\hat{z}$ with magnitude k. Locally, a curved wavefield looks like a plane wave travelling in some direction $\hat{n}$ so that $\nabla\phi = k\hat{n}$. Hence $\hat{n} = \nabla\phi/k$, or $\hat{n}_\perp = \nabla_\perp\phi/k$.

Then $\frac{1}{k}\nabla_\perp \cdot [I(r)\nabla_\perp\phi(r)] = \nabla_\perp \cdot [I(r)\hat{n}_\perp] = -\frac{\partial I(r)}{\partial z}$,
which is the equation governing the GOA (ray picture).

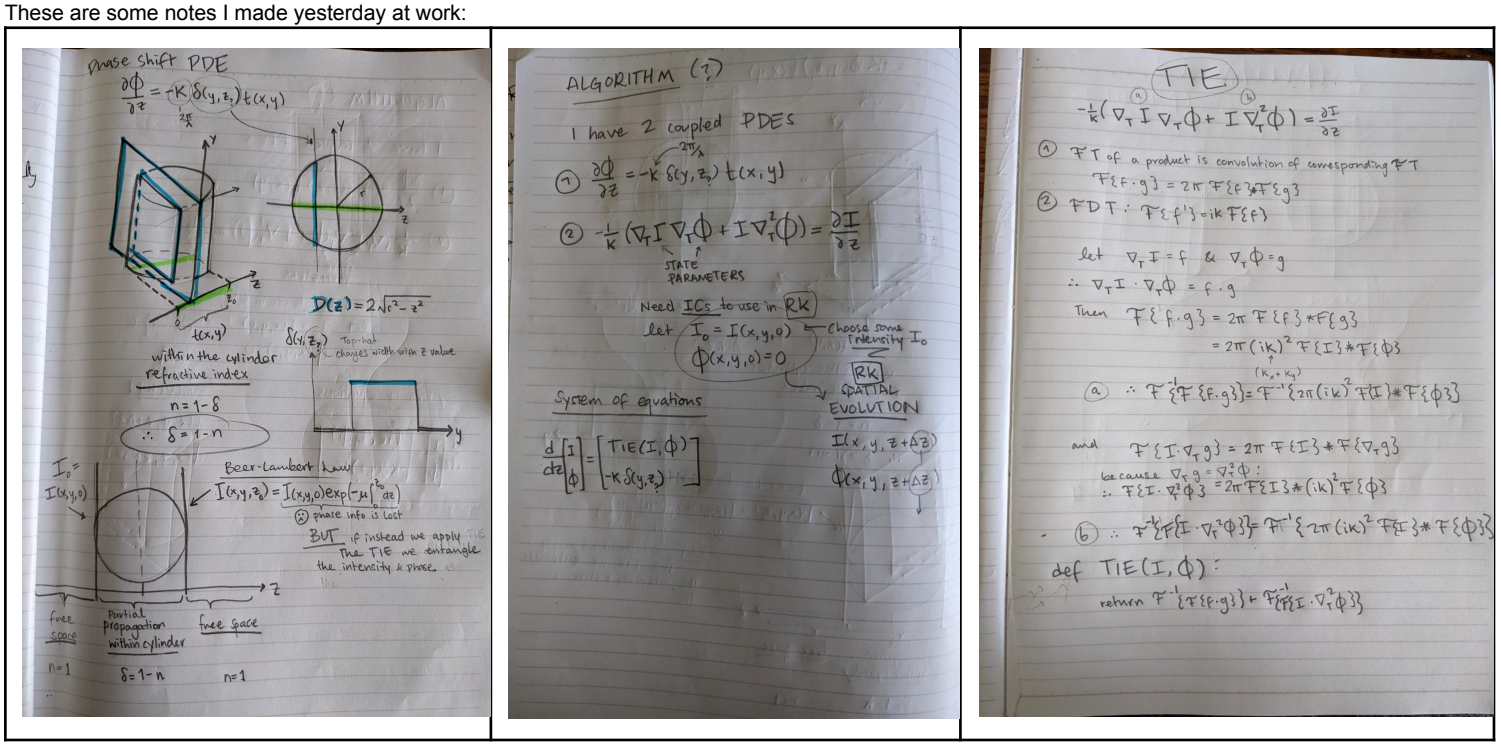


The TIE is for 'near field' diffraction only that doesn't allow for interference effects like Fresnel or Fraunhoffer Diffraction. The fringes are purely geometric and are fixed in position w.r.t. the object.

13

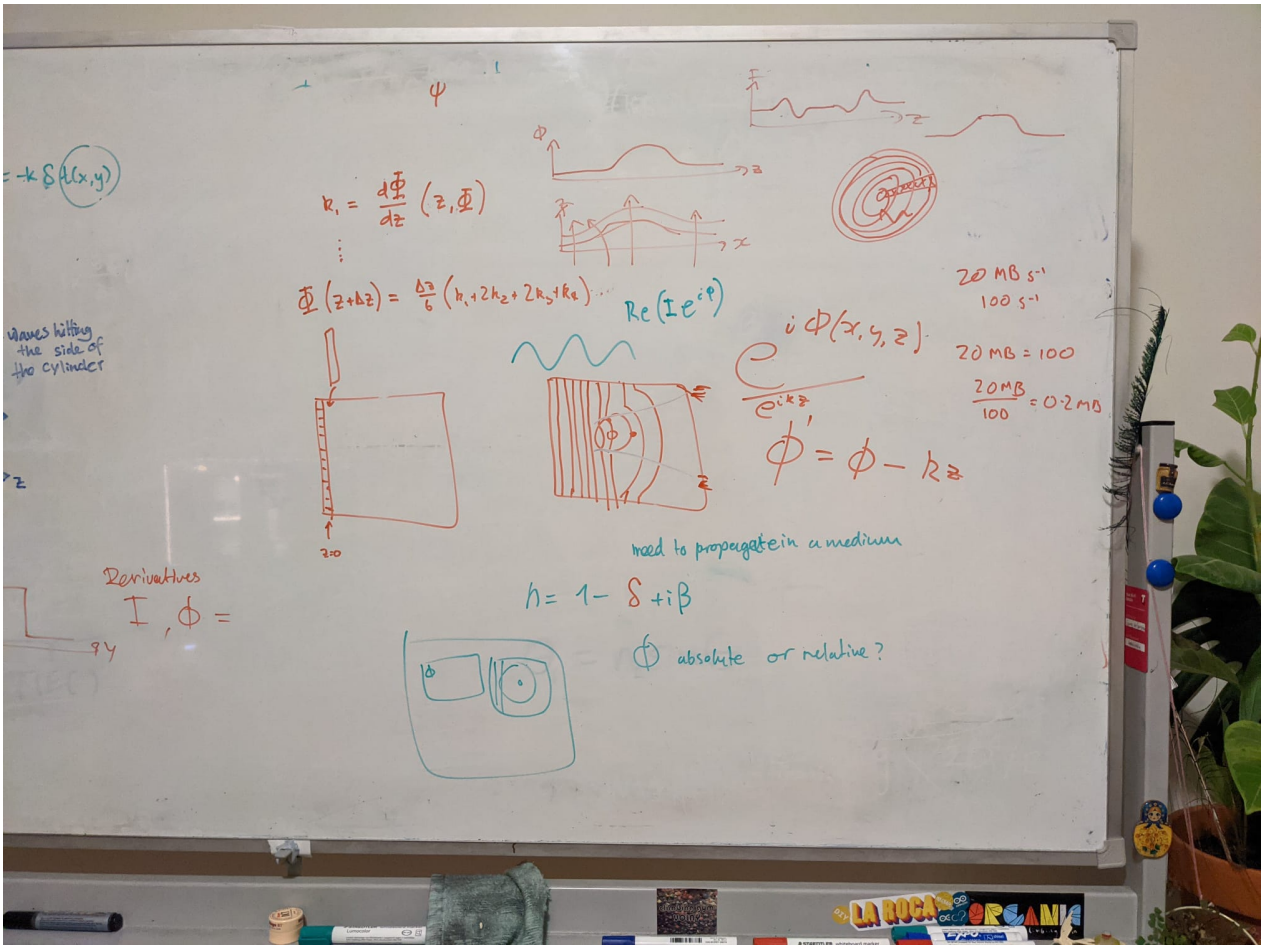
Can I use the Fourier Derivative theorem?
Difference in refractive index dominates phase gradients

06/08/2021
These are some notes I made yesterday at work:



Oops this last line in the right-most image is missing a factor of -1/k

08/08/21
This is a picture of my whiteboard chats with Chris



I've also watched some of Pags' lectures on phase contrast imaging and wow. He is really a great teacher.
A few parts of the theory actually clicked in my brain now… I'm in the process of writing some code at the moment, but I am unsure about how to write the phase equation.

For my complex refractive index I am planning on using the values displayed in (Beltran et al. 6430)

Table 1. Values of δ and μ at 24 keV for materials used to construct the test object in Fig. 2. Values obtained from http://henke.lbl.gov/optical_constants/ (accessed Nov. 4, 2009).

Material	δ ($\times 10^{-7}$)	μ (m^{-1})
PMMA ($C_5H_8O_2$)	4.628	41.2
PTFE (H_2F_4)	7.789	119.8
Aluminium	9.396	502.6

Input interpretation $\frac{hc \text{ (Planck constant-speed of light product)}}{24 \text{ keV (kiloelectronvolts)}}$ Result 5.166×10^{-11} meters	This will be my X-ray wavelength… I chose this value based on the parameters provided in (Beltran et al. 6430) First I will use the δvalue for **PMMA**

Some of the code I have so far:

```python
import os
from pathlib import Path
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import physunits
from scipy.fft import fft, ifft
from physunits import nm

plt.rcParams['figure.dpi'] = 200

folder = Path('TIE')
os.makedirs(folder, exist_ok=True)
os.system(f'rm {folder}/*.png')

# functions

def TIE(z, I, ϕ):
    # propagating in free space
    return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(ϕ), mode='same')))

def dzdϕ(x_array, z_value):
    return - k * δ(x_array, z_value)

def δ(x_array, z_value):
    # constant inside the cylinder but zero everywhere else (i.e. depends on the geometry of the imaged object)
    δ0 = 939.6 * nm
    δ_array = np.zeros_like(x_array)
    D = 2 * np.sqrt(r**2 - z_value**2)
    print((f"{D = }\n"))

    # print(δ_array[x_array == D] + δ0)
```

I am still quite unsure about lines 25 to 32…
I need to think about how to express this function in a way that makes more sense since I am not sure about my usage of the D variable.

Clearly I have to set the value of δto be δ0when the values in the x array are equal to those present in D...The issue here is that D is NOT an array because r (line 45) is not an array! DUH I need to make it be one.

Week 3 (09/08/21 –15/08/21)

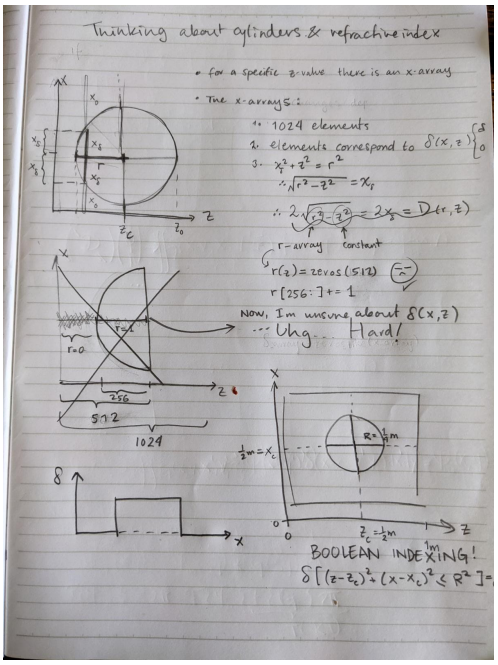
Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Meetings on **Wednesday 11/08**
    a. one-on-one: **11 am**
    b. ~~Group: **1 pm (Didn't attend due to erroneous scheduling)**~~
2. ~~**Fix r in code**~~
3. **Read X-Ray Handbook**
4. **Read PyQt GUI book**
5. ~~Progress report 1 (start figuring out what to write)~~

09/08/2021



Code

Actually after a little while of writing things and later “rubber-ducking” with Chris, I came to the conclusion that I don’t need to use an array for r. I can simply use Boolean indexing on my array of δusing the rule of a circle as logical inequality, and establishing a constant radius R.

In line 21, I assign $\delta_0 = 462.8$nm as the refractive index of the Perspex cylinder matching (Beltran et al. 6430). In line 26, I assign the circle rule to the $\delta(x, z)$array as Boolean inequality indexing.

```
21  def δ(x_array, z_value):
22      # refractive index: constant inside the cylinder but zero everywhere else
23      δ0 = 462.8 * nm
24      δ_array = np.zeros_like(x_array)
25      for i in range(len(x_array)):
26          δ_array[(z_value - z_c)**2 + (x_array[i] - x_c)**2 <= R**2] = δ0
27
```

...

I also define the value R as half of the reported Perspex cylinder diameter d = 12.75

```
    # circle parameters
    R = 12.75 / 2 * m
    z_c = 0.5 * m
    x_c = 0.5 * m

```

The output:

```
δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
       4.628e-07])

δ_array = array([4.628e-07, 4.628e-07, 4.628e-07, ..., 4.628e-07, 4.628e-07,
```

Yay! I think this worked. The δarrays become zero after reaching the end of the “circular cross sectional” region once again.
I now wrote the RK algorithm to do space propagation of my state vector $d\psi\_dz$

```
# functions
def TIE(z, I, Φ):
    # propagating in free space
    return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(Φ), mode='same')))

def δ(x_array, z_value):
    # refractive index: constant inside the cylinder but zero everywhere else
    δ0 = 462.8 * nm
    δ_array = np.zeros_like(x_array)
    for i in range(len(x_array)):
        δ_array[(z_value - z_c)**2 + (x_array[i] - x_c)**2 <= R**2] = δ0
        print(f"{δ_array = }\n")

def dΨ_dz(z, Ψ):
    # state vector of derivatives in z
    I, Φ = Ψ
    dI_dz = TIE(z, I, Φ)
    dΦ_dz = - k * δ(x, z)
    return np.array([dI_dz, dΦ_dz])

def Runge_Kutta(z, delta_z, Ψ):
    # spatial evolution 4th order RK
    k1 = dΨ_dz(z, Ψ)
    k2 = dΨ_dz(z + delta_z / 2, Ψ + k1 * delta_z / 2)
    k3 = dΨ_dz(z + delta_z / 2, Ψ + k2 * delta_z / 2)
    k4 = dΨ_dz(z + delta_z, Ψ + k3 * delta_z)
    return Ψ + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)


if __name__ == '__main__':

    # x- array parameters
    x_max = 5
    x = np.linspace(-x_max, x_max, 1024, endpoint=False)
    n = x.size
    x_step = x[1] - x[0]

    # X-ray beam parameters
    λ = 0.01 * nm # x-rays wavelength
    k0 = 2 * np.pi / λ

    # For Fourier space (one dimension only)
    k = 2 * np.pi * np.fft.fftfreq(n, x_step)
```

```
ana@ana-XPS-13-9343: ~/Desktop/proj2/X-ray-kitchen/code
[ana:code]$ python3 TIE_test.py
rm: cannot remove 'TIE/*.png': No such file or directory
Traceback (most recent call last):
  File "TIE_test.py", line 93, in <module>
    state_vector = Runge_Kutta(z, delta_z, I, Φ)
TypeError: Runge_Kutta() takes 3 positional arguments but 4 were given
[ana:code]$ python3 TIE_test.py
TIE_test.py:19: RuntimeWarning: divide by zero encountered in true_divide
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(Φ), mode='same')))
TIE_test.py:19: RuntimeWarning: invalid value encountered in multiply
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(Φ), mode='same')))
Traceback (most recent call last):
  File "TIE_test.py", line 95, in <module>
    state_vector = Runge_Kutta(z, delta_z, state_vector)
  File "TIE_test.py", line 38, in Runge_Kutta
    k1 = dΨ_dz(z, Ψ)
  File "TIE_test.py", line 33, in dΨ_dz
    dΦ_dz = - k * δ(x, z_value)
NameError: name 'z_value' is not defined
[ana:code]$ python3 TIE_test.py
TIE_test.py:19: RuntimeWarning: divide by zero encountered in true_divide
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(Φ), mode='same')))
TIE_test.py:19: RuntimeWarning: invalid value encountered in multiply
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(Φ), mode='same')))
δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])

δ_array = array([0., 0., 0., ..., 0., 0., 0.])
```

Debugging TIE_test.py

I am getting these errors:

```
[ana:code]$ python3 TIE_test.py
TIE_test.py:19: RuntimeWarning: divide by zero encountered in true_divide
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(ϕ), mode='same')))

TIE_test.py:19: RuntimeWarning: invalid value encountered in multiply
  return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * np.convolve(fft(I), fft(ϕ), mode='same')))
```

I'm not sure about these errors
Probably need to print every item here and see the details.
...

```
Traceback (most recent call last):
  File "TIE_test.py", line 33, in dΨ_dz
    dϕ_dz = - k * δ(x, z)
TypeError: unsupported operand type(s) for *: 'float' and 'NoneType'
```

I think this las error occurs because I didn't write a return statement on δ(x, z)
My fix worked, the error is gone.

divide by zero encountered in true_divide & invalid value encountered in multiply errors:

```
16  # functions
17  def TIE(z, I, ϕ):
18      # propagating in free space
19      print(f"{- (1 / k) = }\n")
20
21      print(f"{np.convolve(fft(I), fft(ϕ)) = }\n")
22
23      print(f"{2 * ifft(- 2 * np.pi * k**2 * np.convolve(fft(I), fft(ϕ)))}\n")
24
25      assert 0
26
```

```
[ana:code]$ python3 TIE_test.py
TIE_test.py:19: RuntimeWarning: divide by zero encountered in true_divide
  print(f"{- (1 / k) = }\n")
- (1 / k) = array([       -inf, -1.59154943, -0.79577472, ...,  0.53051648,
        0.79577472,  1.59154943])

np.convolve(fft(I), fft(ϕ)) = array([0.+0.j, 0.+0.j, 0.+0.j, ..., 0.+0.j, 0.+0.j, 0.+0.j])

Traceback (most recent call last):
  File "TIE_test.py", line 104, in <module>
    Ψ = Runge_Kutta(z, delta_z, Ψ)
  File "TIE_test.py", line 47, in Runge_Kutta
    k1 = dΨ_dz(z, Ψ)
  File "TIE_test.py", line 41, in dΨ_dz
    dI_dz = TIE(z, I, ϕ)
  File "TIE_test.py", line 23, in TIE
    print(f"{2 * ifft(- 2 * np.pi * k**2 * np.convolve(fft(I), fft(ϕ)))}\n")
ValueError: operands could not be broadcast together with shapes (1024,) (2047,)
```

Yup. Several errors but let's start from the top. The value -1/k -> -∞

1. This is how I defined **k**

```
66      # For Fourier space (one dimension only)
67      k = 2 * np.pi * np.fft.fftfreq(n, x_step)
68
```

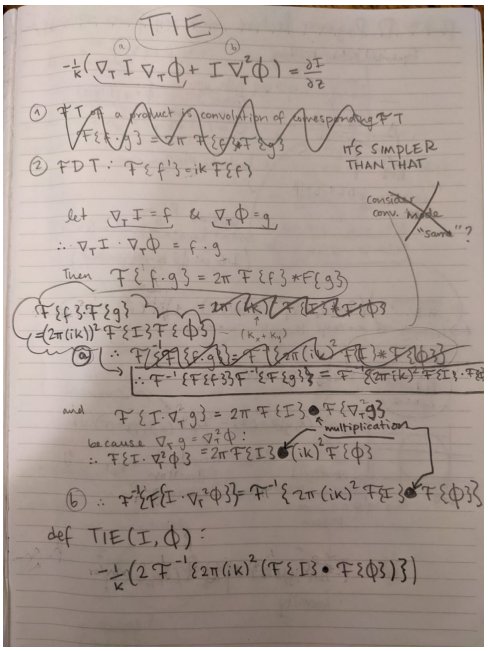
The returned float array contains the frequency bin centers in cycles per unit of the sample spacing (**with zero at the start**) (The NumPy community).

Maybe I have to use a mask on the invalid values

2. ~~The convolution does get calculated.. HOPEFULLY as the algorithm evolves the state vector this convolution makes more sense than only a big array of zeros.~~
   ~~**I need to check that my TIE actually makes sense**~~
3. There is a failed broadcast between **k** and the **ifft(...) I reckon that this is likely due to the convolution.**

```
(k**2).shape = (1024,)

np.convolve(fft(I), fft(ϕ), mode='full').shape = (2047,)
```

Chris told me that I don't need to use a convolution and instead can simply multiply the respective FTs



This is my new expression:

```
16  # functions
17  def TIE(z, I, ϕ):
18      # propagating in space
19      return - (1 / k) * (2 * ifft(2 * np.pi * (- k**2) * fft(I) * fft(ϕ)))
20
```

I still need to fix the issue stemming from the `-1/k` array

10/08/21
Ok so I slept on it and my TIE error was simply that I was using the wrong **k (!)**
I have now changed the k (fourier wave vector) to k0 the X-ray wavenumber.

```
16  # functions
17  def TIE(z, I, ϕ):
18      # propagating in space
19      return - (1 / k0) * (2 * ifft(2 * np.pi * (- k**2) * fft(I) * fft(ϕ)))
```

```
55      # X-ray beam parameters
56      λ = 0.01 * nm # x-rays wavelength
57      k0 = 2 * np.pi / λ
58
59      # For Fourier space (x dimension only)
60      k = 2 * np.pi * np.fft.fftfreq(n, x_step)
```

These errors are no more! YAY!

Testing δ(x, z)

```
102     ### PLAYING AROUND with δ ###
103     x = np.linspace(-5, 5, 1024, endpoint=False)
104     z = np.linspace(0, 1, 1000, endpoint=False).reshape((1000, 1))
105     δ_array = δ(x, z)
106     plt.imshow(δ_array)
107     plt.show()
108     ### ------------ ###
```

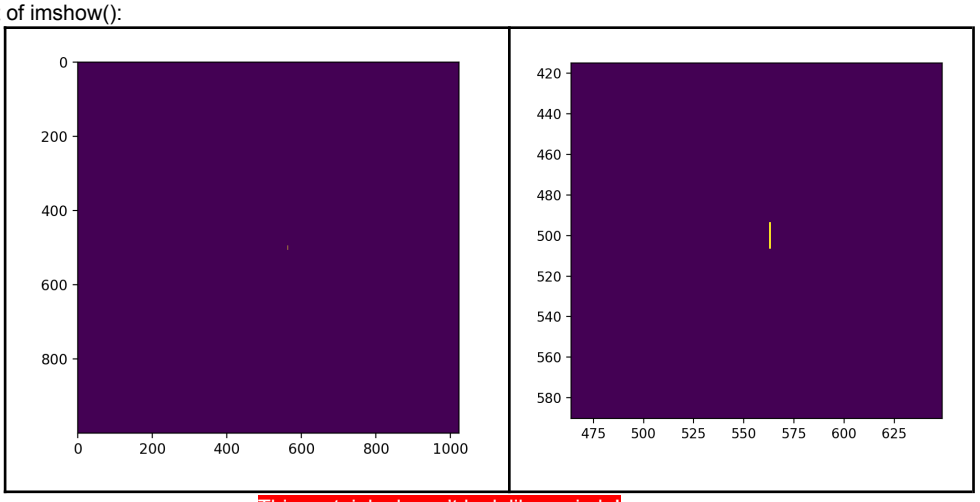
Output:

```
[ana:code]$ python3 TIE_test.py
Traceback (most recent call last):
  File "TIE_test.py", line 82, in <module>
    δ_array = δ(x, z)
  File "TIE_test.py", line 27, in δ
    δ_array[(z_value - z_c) ** 2 + (x_array[i] - x_c) ** 2 <= R ** 2] = δ0
IndexError: boolean index did not match indexed array along dimension 0; dimension is 1024 but corresponding boolean dimension is 1000
[ana:code]$
```

To fix this I need to reshape z in order to make δ(x, z) a 2D array to be visualised with imshow()
`z = np.linspace(0, 1, 1000, endpoint=False).reshape((1000, 1))`
After this I correct my initialised array to actually be a 2D shape using the shape of a “matrix product”

```
22  def δ(x, z):
23      # refractive index: constant inside the cylinder but zero everywhere else
24      δ0 = 462.8 * nm
25      δ_array = np.zeros_like(x * z)
26      δ_array[(z - z_c) ** 2 + (x - x_c) ** 2 <= R ** 2] = δ0
27      # print(f"\n{np.shape(δ_array) = }\n")
28      return δ_array
```

This is the visual output of imshow():



This certainly doesn't look like a circle!

11/08/21

Debugging δ(x, z)

Investigating if my imshow() plot is correctly proportioned

Currently my x array is `x = np.linspace(-x_max, x_max, 1024, endpoint=False)`

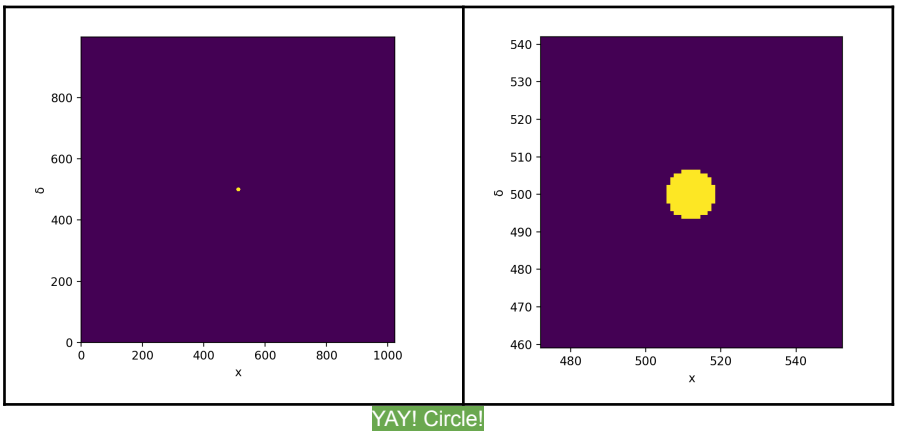
where `x_max = 5,`

and my z array is `z = np.linspace(0, 1, 1000, endpoint=False).reshape((1000, 1))`

Maybe because these arrays have different spans this could be causing me to observe a stretched "circle"?

```
102     ### PLAYING AROUND with δ ###
103     x = np.linspace(0, 1, 1024, endpoint=False)
104     z = np.linspace(0, 1, 1000, endpoint=False).reshape((1000, 1))
105     δ_array = δ(x, z)
106     plt.imshow(δ_array, origin='lower')
107     plt.xlabel("x")
108     plt.ylabel("δ")
109     plt.show()
110     ### ------------ ###
```

Output:



YAY! Circle!

MEETING DAY!

Comments and Notes (tasks) from my chat with MK:

1. *Using the **projection approximation** might give me an easier set of ICs for my TIE & spatial evolution*

$$I(x, y, z_0) = \exp\left[-\mu t(x, y)\right] I(x, y, 0),$$

   I think that maybe there were a few misunderstandings in the meeting.
   I might have to send an email to MK to verify my interpretations of his suggestions.

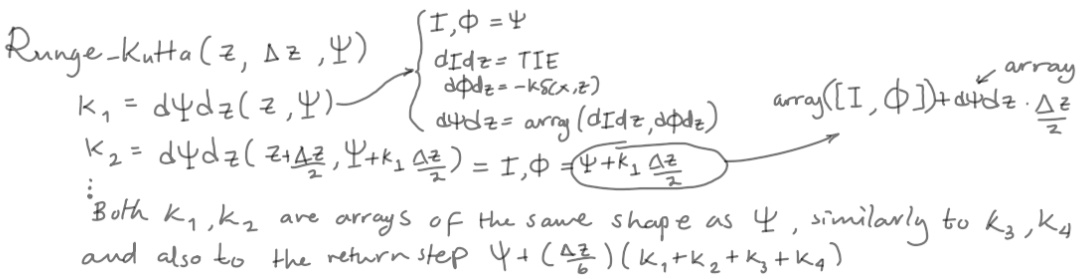
2. ~~*Focus on drawing a cylinder in 3D space*~~
   a. ~~*MAKE SURE IT IS A SOLID OBJECT NOT A SURFACE*~~
3. *Write a detailed plan for **report 1** and send it to MK*
   a. ***Title: How material density affects phase contrast Imaging***

...continuing my work as it is for now..

```python
def Runge_Kutta(z, delta_z, Ψ):
    # spatial evolution 4th order RK
    k1 = dΨ_dz(z, Ψ)
    k2 = dΨ_dz(z + delta_z / 2, Ψ + k1 * delta_z / 2)
    k3 = dΨ_dz(z + delta_z / 2, Ψ + k2 * delta_z / 2)
    k4 = dΨ_dz(z + delta_z, Ψ + k3 * delta_z)
    return Ψ + (delta_z / 6) * (
        k1 + 2 * k2 + 2 * k3 + k4
    )  # what is this doing????? Ψ + ___ ???

```

In my previous work (with Jesper and Meera) I've only used a single state variable in RK. Therefore here I must rewrite my algorithm in the context of my current problem with I and ϕas the state variables of the state vector Ψ.

```python
def Runge_Kutta(z, delta_z, Ψ):
    # spatial evolution 4th order RK
    # z is single value
    # Ψ is array with shape: (2, 2048)
    # print(f"\n{np.shape(Ψ) = }\n")

    k1 = dΨ_dz(z, Ψ) # array
    # print(f"\n{np.shape(k1) = }\n")

    k2 = dΨ_dz(z + delta_z / 2, Ψ + k1 * delta_z / 2) # array
    # print(f"\n{np.shape(k2) = } \n")

    k3 = dΨ_dz(z + delta_z / 2, Ψ + k2 * delta_z / 2) # array
    # print(f"\n{np.shape(k3) = }\n")

    k4 = dΨ_dz(z + delta_z, Ψ + k3 * delta_z) # array
    # print(f"\n{np.shape(k4) = }\n")

    # print(f"\n{np.shape(Ψ + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)) = }\n")
    return Ψ + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4) # array shape (2, 2048)

```




```
[ana:code]$ python3 TIE_test.py
i = 0

np.shape(Ψ) = (2, 2048)


np.shape(k1) = (2, 2048)


np.shape(k2) = (2, 2048)


np.shape(k3) = (2, 2048)


np.shape(k4) = (2, 2048)


np.shape(Ψ + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)) = (2, 2048)
```

This outcome makes things more tricky because now I am unsure of what is going wrong. Given that my RK algorithm is apparently working as expected.

¿What is wrong with my algorithm?

If state vector is $\Psi = \begin{bmatrix} I \\ \phi \end{bmatrix}$

Taking the derivative of Ψ wrt z:

$$\therefore \frac{\partial \Psi}{\partial z} = \begin{bmatrix} \frac{\partial I}{\partial z} \\ \frac{\partial \phi}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{-1}{k_0}(\nabla_T I \nabla_T \phi + I \nabla_T^2 \phi) \\ -k_0 \delta(x,z) \end{bmatrix}$$

where:
- $k_0 = 2\pi/\lambda$
- K = 2π·fftfreq(n, x_step)

previously I had K = (fourier space k) which was a typo

By FDT: for any continuous, differentiable f, g:

$$\mathcal{F}\{\nabla f \nabla g\} + \mathcal{F}\{f \nabla^2 g\} = 2\pi(ik)\mathcal{F}\{f\} + 2\pi(ik_T)\mathcal{F}\{g\} + 2\pi\mathcal{F}\{f\} + 2\pi(ik)^2 \mathcal{F}\{g\}$$

TIE(z, I, ϕ):

$$\therefore \frac{-1}{k_0}(\nabla_T I \nabla_T \phi + I \nabla_T^2 \phi) = \frac{-1}{k_0}\left(\mathcal{F}^{-1}\{2\pi(ik_T)\mathcal{F}\{I\}\, 2\pi(ik_T)\mathcal{F}\{\phi\}\} + \mathcal{F}^{-1}\{2\pi\mathcal{F}\{I\}\, 2\pi(-k_T^2)\mathcal{F}\{\phi\}\}\right)$$

where $k_T = (k_x, k_y)$ previously this K was single dimensional (on x)

$$\therefore TIE(z, I\ \phi) = \frac{-1}{k_0}\left(\mathcal{F}^{-1}\{4\pi^2(-k_T^2)\mathcal{F}\{I\}\mathcal{F}\{\phi\}\} + \mathcal{F}^{-1}\{4\pi^2(-k_T^2)\mathcal{F}\{I\}\mathcal{F}\{\phi\}\}\right)$$

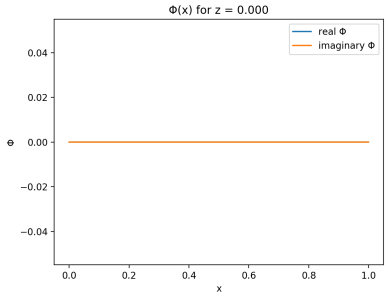
$$= \frac{-1}{k_0}\left(2\mathcal{F}^{-1}\{4\pi^2(-k_T^2)\mathcal{F}\{I\}\mathcal{F}\{\phi\}\}\right)$$

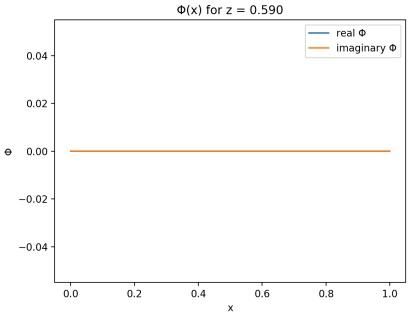
So far I've found a typo in my maths, I was missing a factor of 2π.
Should I change k to k_perp eventually*?
After correcting this error:
Nothing changes in the test plots of Φvs x

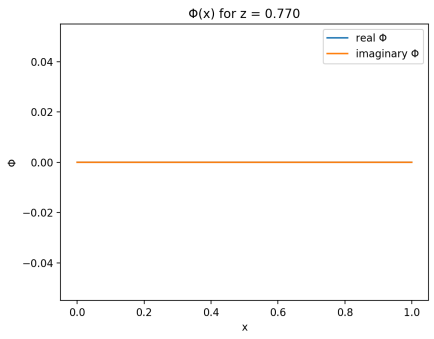
```
97      while z < z_final:
98
99          print(f"{i = }")
100
101         # # TEST PLOT
102         # zig zag ??
103         if not i % 10:
104             plt.plot(x, np.real(Φ), label="real Φ")
105             plt.plot(x, np.imag(Φ), label="imaginary Φ")
106             plt.xlabel("x")
107             plt.ylabel("Φ")
108             plt.legend()
109             plt.title(f"Φ(x) for z = {z:.03f}")
110             plt.savefig(folder/f'z ={i:04d}.png')
111             # plt.show()
112             plt.clf()
113
114         # spatial evolution step
115         Ψ = Runge_Kutta(z, delta_z, Ψ)
116         i += 1
117         z += delta_z
```

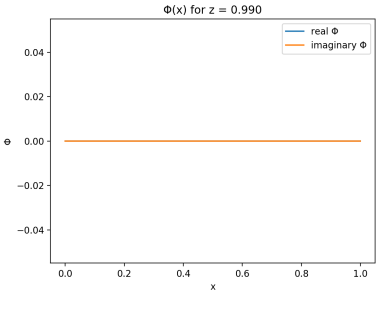
Here **I would expect** some phase gained or lost as the algorithm evolves the state in the z direction, specially in the region that has a decreased refractive index: $n = 1 - \delta(x, z)$i.e. the cylinder cross section (assigned in my $\delta(x, z)$function)
The phase remains as it was set in the initial conditions for the state vector

```
# ICs
    I = np.ones_like(x)
    Φ = np.zeros_like(x)
    # Initial state vector
    Ψ = np.array([I, Φ])
```









Debugging dΨ_dz(z, Ψ)

Since I've corrected my TIE(z, I, ϕ) and δ(x, z) functions the next step is to test and debug the state vector function:

```
29  def dΨ_dz(z, Ψ): #could construct a z and Ψ and check with them in isolation from RK
30      # state vector of derivatives in z
31      I, Φ = Ψ
32      dI_dz = TIE(z, I, Φ)
33      dΦ_dz = -k0 * δ(x, z) # how much should this grow per z?
34      # plot
35      return np.array([dI_dz, dΦ_dz])
```

I want to verify how much dΦ_dz = -k0 * δ(x, z) actually grows as RK spatially evolves the state in z.
To do this I need to first modify the distance parameter x_max,
(I also use x_max as the upper propagation limit in the z-direction array).

```
87      # x- array parameters
88      x_max = 1 * m # make this smaller!
89      x = np.linspace(0, x_max, 2048, endpoint=False)
90      n = x.size
91      x_step = x[1] - x[0]
```

```
103     # Propagation & loop parameters
104     i = 0
105     z = 0 * m
106     z_final = x_max
107     delta_z = 0.001 * m
```

Since the refractive region defined by the cylindrical cross section is actually much smaller than the propagation distance I made up (i.e. t(x, z) = 12.75 mm vs 1 m) maybe this is why I am not seeing any phase change as I propagate in z.
I also want to change the position of the cylinder in space, from:

```
102     # Cylinder parameters
103     R = 12.75 / 2 * mm
104     z_c = 0.5 * m # move to zero
105     x_c = 0.5 * m # move to zero and set x_max and min to be sym about zero
```

To zero to have axial symmetry along x.

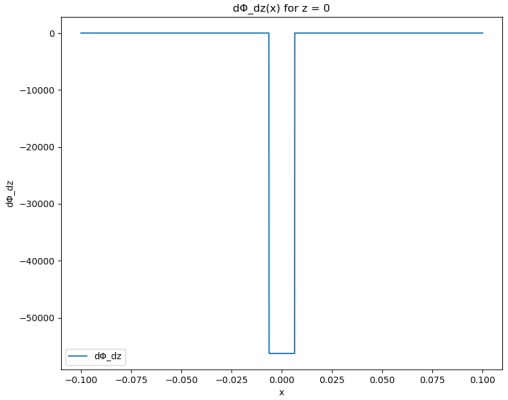
Code changes

```
83      # x-array parameters
84      x_max = 100 * mm
85      x_min = - 100 * mm
86      x = np.linspace(x_min, x_max, 2048, endpoint=False)
87      n = x.size
88      x_step = x[1] - x[0]
```

```
 97     # Propagation & loop parameters
 98     i = 0
 99     z = 0 * mm
100     z_final = x_max
101     delta_z = 1 * mm
102
103     # Cylinder parameters
104     R = 12.75 / 2 * mm
105     z_c = 0 * mm
106     x_c = 0 * mm
```

```
37  def dΨ_dz(z, Ψ):
38      # state vector of derivatives in z
39      I, Φ = Ψ
40      dI_dz = TIE(z, I, Φ)
41      dΦ_dz = -k0 * δ(x, z)
42      # print(f"\n{np.shape(dΦ_dz[:]) = }\n") # this returns: np.shape(δ_array) = (2048,)
43      # test PLOT #
44      # how much should this grow per z?
45      plt.plot(x, dΦ_dz[:], label="dΦ_dz")
46      plt.xlabel("x")
47      plt.ylabel("dΦ_dz")
48      plt.legend()
49      plt.title(f"dΦ_dz(x) for z = {z}")
50      plt.savefig(folder/f'z ={z}.png')
51      # plt.show()
52      plt.clf()
```

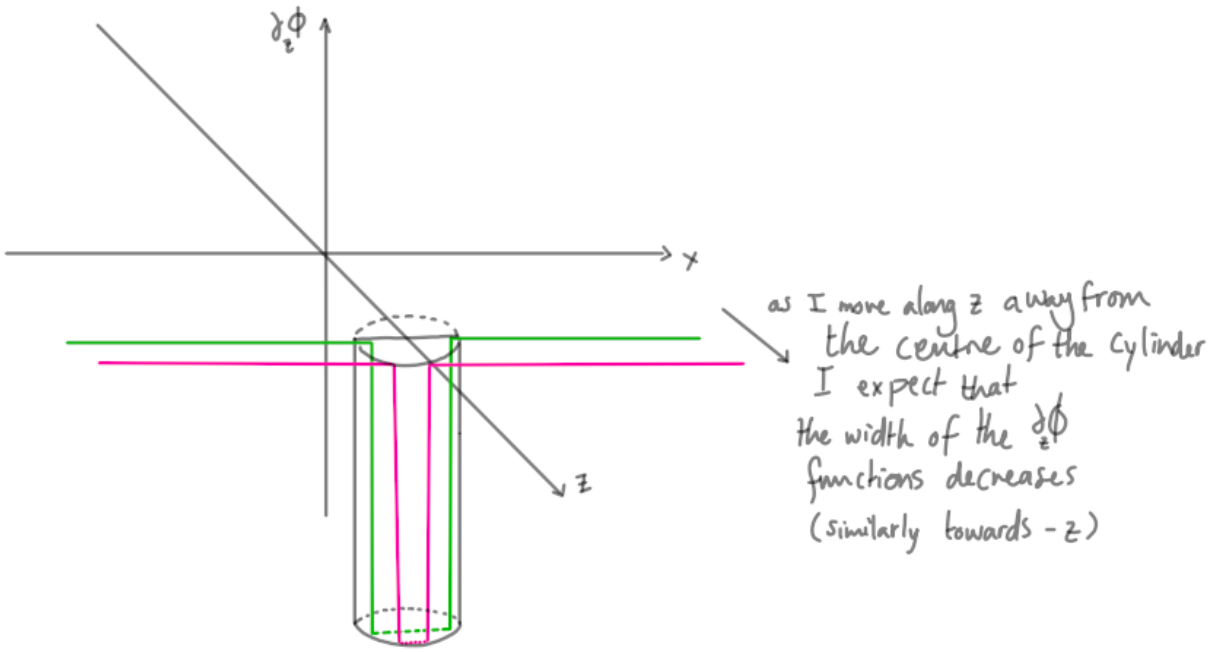
Output:

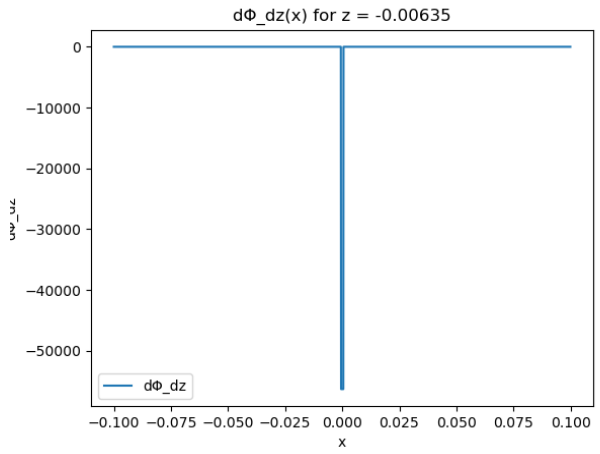


Neat! This looks like there is some phase shift going on in the area where the object is.

15/08/21

I should check this for more values of z. Theoretically this inverted top hat function should decrease width as z moves away from 0 in either direction (due to the radial symmetry of the distribution)

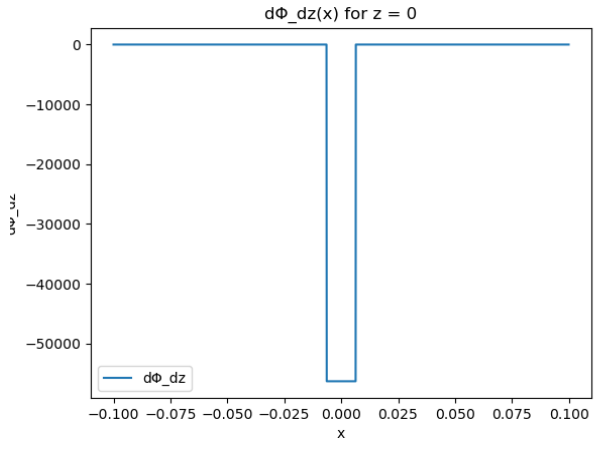


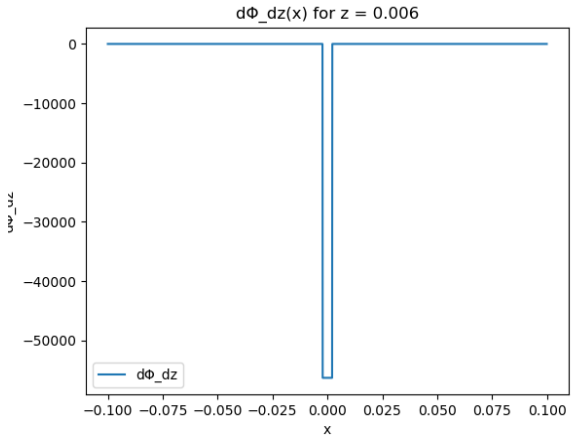


The change in phase wrt z behaves as expected!

This probably means that the fact that I am not able to observe any change in phase after the RK algorithm must be because the issues in my code might be occurring **while** in the RK evolution loop.

Therefore, I will test my dΦ_dz function within the RK loop





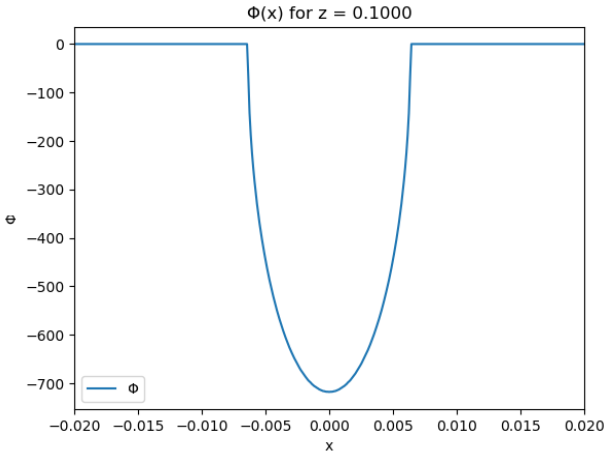
I noticed that I had set the range of z incorrectly … I previously had z = (0, 1)
But that made no sense given that I centered the cylinder at (x, z) = (0, 0)
I also change the step size of the spatial evolution in z: delta_z to 0.1mm

```
95      # Propagation & loop parameters
96      i = 0
97      z = -x_max * mm # I changed this to match the symmetry of the cylinder instead of starting in the middle
98      z_final = x_max
99      delta_z = 0.1 * mm
```

Runge Kutta evaluation

```
    ################## evolution algorithm ##########################

    # ICs
    I = np.ones_like(x)
    Φ = np.zeros_like(x)
    # Initial state vector
    Ψ = np.array([I, Φ])

    while z < z_final:

        # I unpack the state vector to visualise the phase
        I, Φ = Ψ
        # test PLOT #
        if i in range(9):
            plt.plot(x, Φ[:], label="Φ")
            plt.xlim(-20 * mm, 20 * mm)
            plt.xlabel("x")
            plt.ylabel("Φ")
            plt.legend()
            plt.title(f"Φ(x) for z = {z:.4f} inside RK loop")
            plt.savefig(folder/f'z ={z:.4f}.png')
            # plt.show()
            plt.clf()

        if not i % 10:
            plt.plot(x, Φ[:], label="Φ")
            plt.xlim(-20 * mm, 20 * mm)
            plt.xlabel("x")
            plt.ylabel("Φ")
            plt.legend()
            plt.title(f"Φ(x) for z = {z:.4f} inside RK loop")
            plt.savefig(folder/f'z ={i:4d}.png')
            # plt.show()
            plt.clf()

        print(f"{i = }")
        # spatial evolution step
        Ψ = Runge_Kutta(z, delta_z, Ψ)
        i += 1
        z += delta_z

    # After the integration occurs I unpack the state vector
    I, Φ = Ψ
```

The phase at the final z value:



Week 4 (16/08/21 –22/08/21)

Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Meetings on **Monday 16/08** and **Wednesday 18/08**
    a. ~~Group: **2 pm (Didn't attend due to Covid vaccine recovery)**~~
    b. ~~one-on-one: **11 am**~~
2. **Read X-Ray Handbook**
3. **Read PyQt GUI book**
4. ~~Send progress report summary to MK~~
5. ~~Submit progress report~~

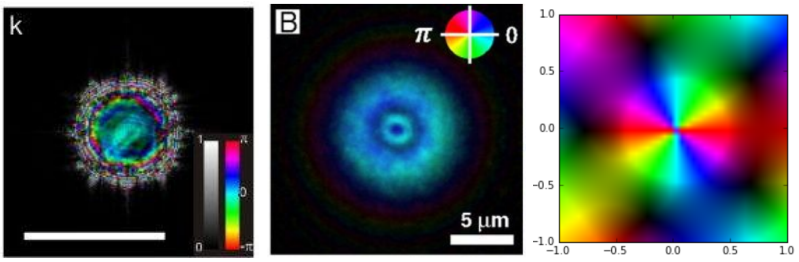
17/08/2021
Since I have verified that my code works, I want to continue with my visualisation planning...
My spatial evolution algorithm returns a (2, 2048) array

$$\Psi = \begin{bmatrix} I \\ \phi \end{bmatrix}$$

I want to make a HSV plot where the color varies as a function of *the state* variables (i.e. hue = phase, saturation = 1, value = intensity)

While looking online for a way to represent phaser and intensity in a single imshow() plot I found this:
https://stackoverflow.com/is-there-any-way-to-use-bivariate-colormaps-in-matplotlib/

Here are some picture examples from Stackoverflow:



Chris's advice:
Note: phase display with imshow

1. Normalise phase:
   phase = phase %2pi
2. Parameter in imshow call:
   cmap = gist_rainbow
3. Hsv channels

18/08/21

MEETING DAY!

Questions for MK:

1. Comments for Report 1:
   a. ~~*Speckle analysis explanation*~~
   b. *Explain the target situation more thoroughly:*
      i. *Peaks and fringes*
      ii. *Characteristics X-rays*
      I decided against including this part in my future plans section
      The reasons for this are
      1. My report is very long already
      2. I might not do this part given the likelihood of covid restrictions extending
   c. ~~*Add Beltran reference*~~

2. Other comments regarding the simulation:
   a. *Things can go wrong in Fourier space, consider:*
      i. *Sanity check with normal derivatives (finite differences)*
      ii. *Deriv function in np*

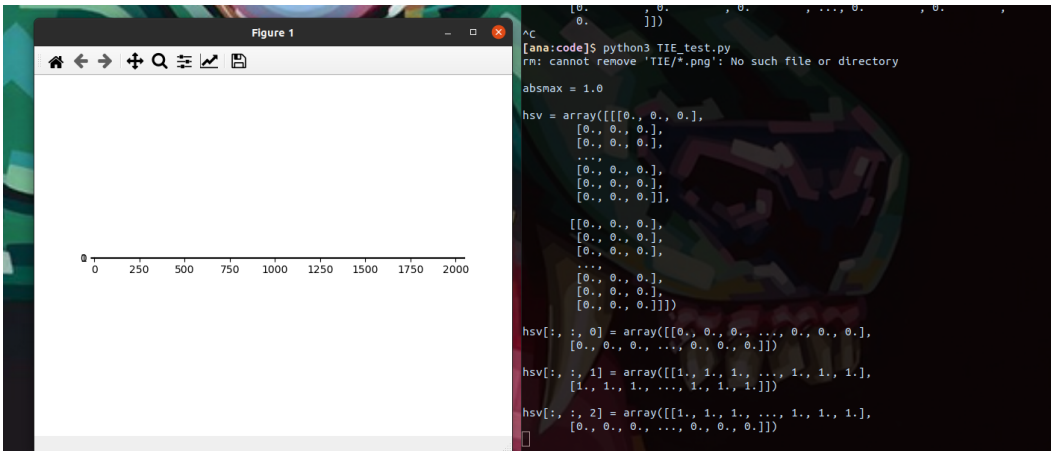
I wrote this code heavily inspired by the code posted in the Stackoverflow link I found yesterday:

```
48  def complex_array_to_rgb(Ψ, i): #, rmax=None):
49      '''Takes an array of complex numbers and converts it to an array of [r, g, b],
50      where phase gives hue and saturaton/value are given by the absolute value.
51      Especially for use with imshow for complex plots.'''
52
53      I, Φ = Ψ
54
55      absmax = np.abs(Ψ).max()
56      print(f"\n{absmax} = ")
57
58      hsv = np.zeros(Ψ.shape + (3,), dtype='float')
59      print(f"\n{hsv} =")
60      hsv[:, :, 0] = Φ / (2 * np.pi) % 1
61      print(f"\n{hsv[:, :, 0] = }")
62
63      hsv[:, :, 1] = 1
64      print(f"\n{hsv[:, :, 1] = }")
65
66      hsv[:, :, 2] = np.clip(np.abs(Ψ) / absmax, 0, 1) # I/I.max() #
67      print(f"\n{hsv[:, :, 2] = }")
68
69      rgb = matplotlib.colors.hsv_to_rgb(hsv)
70      return rgb
71
```

Function call test before the RK loop:

```
109     plt.imshow(complex_array_to_rgb(Ψ, i), cmap="gist_rainbow", origin='lower')
110     # plt.savefig(folder/f'{i:04d}.png')
111     plt.show()
112     plt.clf()
```

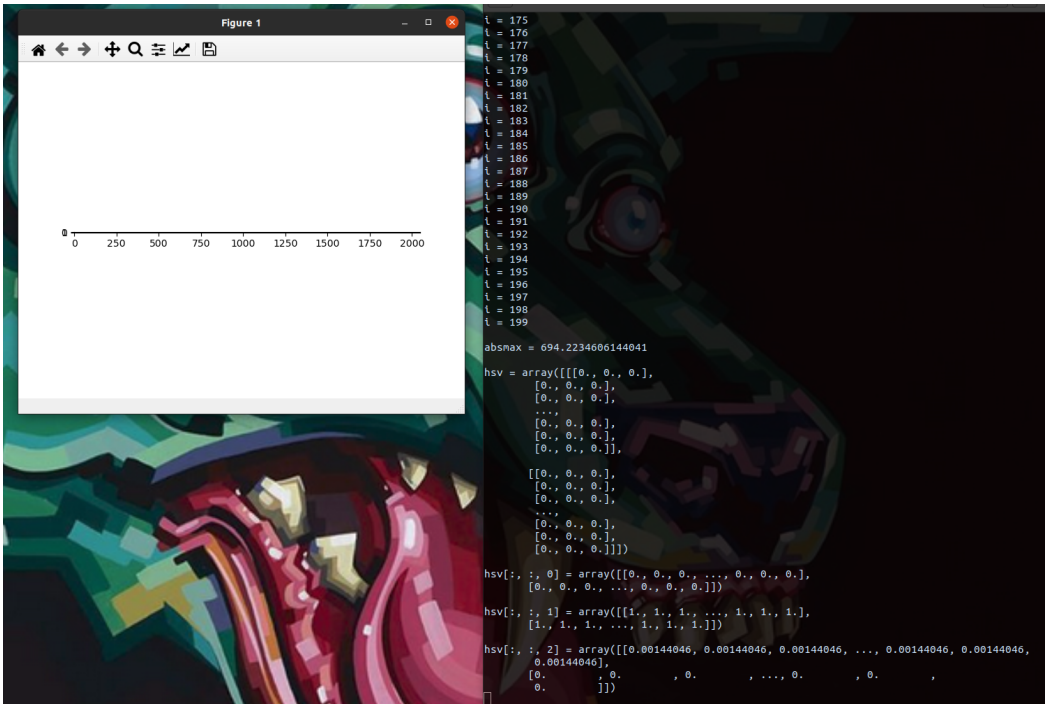
Output:



Function call test after the RK loop

```
125     # Simulation frames
126     plt.imshow(complex_array_to_rgb(Ψ, i), cmap="gist_rainbow", origin='lower')
127     # plt.savefig(folder/f'{i:04d}.png')
128     plt.show()
129     plt.clf()
```

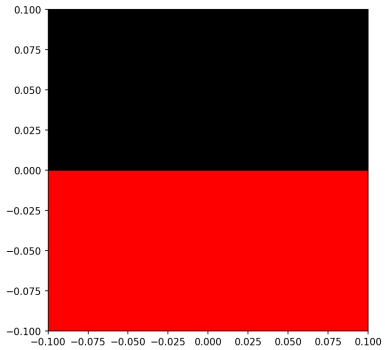
Output:



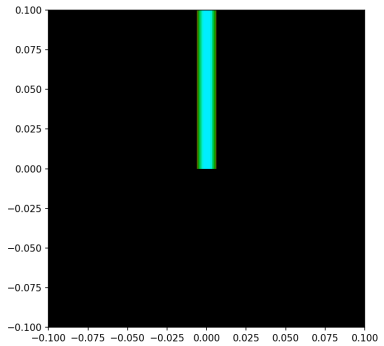
Clearly the spatial evolution is occurring but the plotting function is not being implemented correctly.
Maybe the range is a bit off?
I added **`extent=`** as an extra parameter to the function call, this is the call prior to the RK loop:

```
109     plt.imshow(complex_array_to_rgb(Ψ, i), cmap="gist_rainbow", origin='lower', extent=(-x_max, x_max, -x_max,x_max))
110     # plt.savefig(folder/f'{i:04d}.png')
111     plt.show()
112     plt.clf()
```

Nice, this is a different BUG. Yeah, progress!

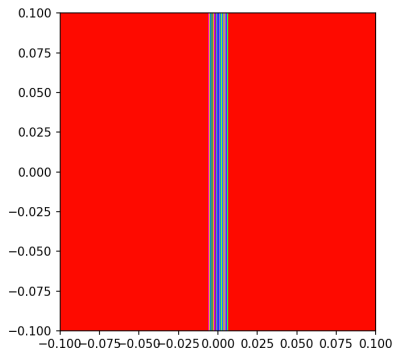
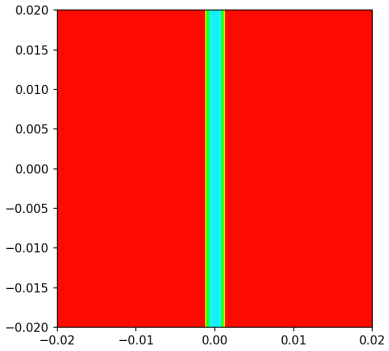


This is the plot output after the RK loop given the same `extent=` parameter added:



Ok I think I identified an issue with line 55, I will use this line 56 instead:

```
55      # absmax = np.abs(Ψ).max()
56      absmax = np.abs(I).max()
57      # print(f"\n{absmax = } ")
```

	This also looks like progress! I think that I can see some phase? I will decrease the extent range to 0.2 * x_max for every direction, `(i.e. extent=(-0.2 * x_max, 0.2 * x_max, -0.2 * x_max, 0.2 * x_max))` For the purpose of saving time I have changed the z-step size to `delta_z = 1 * mm`
	**Ok so, apparently going for the 2D plot presently makes no sense according to Chris.** **He made me see that I am trying to plot 1D data as 2D data and so nopey nope nope… that's pretty stupid.** **I need to rethink things… whoops, but yay progress!**

Current code state:

My globals are:

```python
if __name__ == '__main__':

    # x-array parameters
    x_max = 50 * mm
    x_min = - x_max
    x = np.linspace(x_min, x_max, 2048, endpoint=False)
    n = x.size
    x_step = x[1] - x[0]

    # X-ray beam parameters
    λ = 0.05166 * nm  # x-rays wavelength
    k0 = 2 * np.pi / λ  # x-rays wavenumber

    # Cylinder parameters
    R = 12.75 / 2 * mm
    z_c = 0 * mm
    x_c = 0 * mm

    # For Fourier space
    k = 2 * np.pi * np.fft.fftfreq(n, x_step)

    # Propagation & loop parameters
    i = 0
    z = - x_max
    z_final = x_max
    delta_z = 0.01 * mm # 0.01 * mm # 1 * mm ###### not many steps atm (n_z = 10000)
```

The RK algorithm:

```python
    ######################### evolution algorithm ##############################

    # ICs
    I = np.ones_like(x)
    Φ = np.zeros_like(x)

    Ψ = np.array([I, Φ])

    psi_list = []
    while z < z_final:

        print(f"{i = }")
        # spatial evolution step
        Ψ = Runge_Kutta(z, delta_z, Ψ)
        # print(f"\n{Ψ = }")

        psi_list.append(Ψ)
        i += 1
        z += delta_z

    psi_list = np.array(psi_list) ########### THIS IS NEW
    # print(f"\n{np.shape(psi_list) = }") #(n_z, 2, n_x) = (n_z, 2, 2048)
```

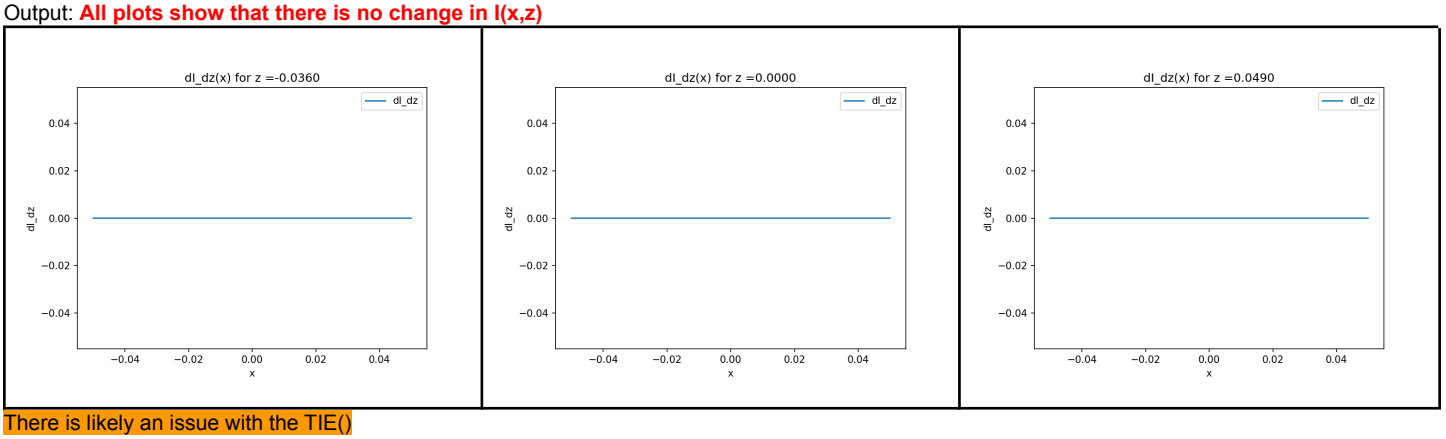
To start: I'm gonna check things out from the top.

1. **In dΨ_dz:**
    a. **dI_dz = TIE()**
        i. **What does this return?**

```python
def dΨ_dz(z, Ψ):
    # state vector of derivatives in z
    I, ϕ = Ψ
    dI_dz = TIE(z, I, ϕ) # how much should this grow per z?
    # print(f"\n{np.shape(dI_dz) = }") #this returns:
    ## Test plot
    if not i % 100:
        plt.plot(x, dI_dz, label="dI_dz")
        plt.xlabel("x")
        plt.ylabel("dI_dz")
        plt.legend()
        plt.title(f"dI_dz(x) for {z =:.4f}")
        plt.savefig(folder/f"{i:4d}")
        # plt.show()
        plt.clf()
        ############
    dϕ_dz = -k0 * δ(x, z)
    return np.array([dI_dz, dϕ_dz])
```

19/08/21

Output: **All plots show that there is no change in I(x,z)**



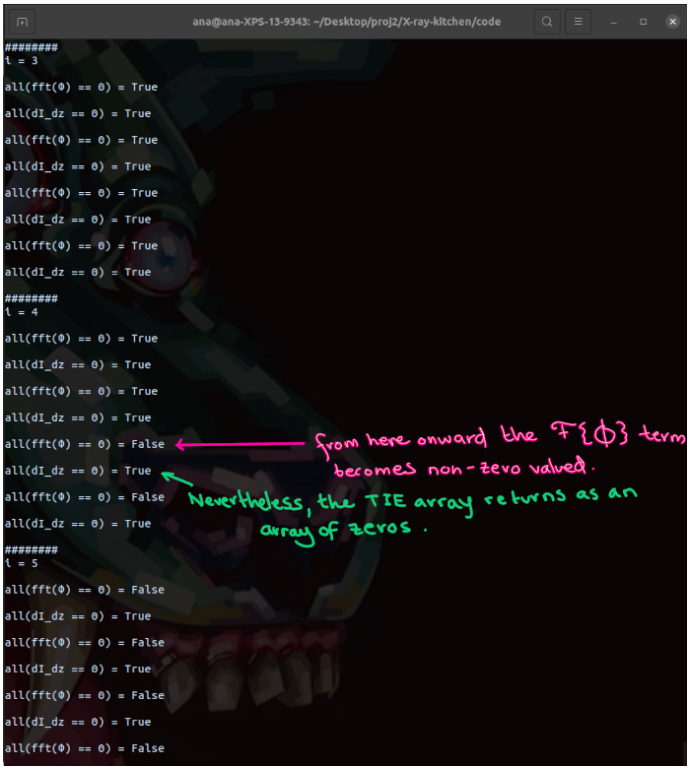
There is likely an issue with the TIE()

I will take that equation apart and look at it term-by-term, since I have obtained a reasonably looking profile for the phase-shift, I expect that the terms with ϕin them should return something nonzero, to verify that ϕis in fact something non zero (which would result in dI_dz = 0)

I've written this code:

```python
def TIE(z, I, ϕ):
    # The intensity and phase evolution of a paraxial monochromatic
    # scalar electromagnetic wave on propagation
    dI_dz = (-1 / k0) * (2 * ifft(4 * np.pi ** 2 * (-k ** 2) * fft(I) * fft(ϕ)))
    print(f"\n{all(fft(ϕ) == 0) = }")
    # print(f"\n{all(fft(I) == 0) = }")
    print(f"\n{all(dI_dz == 0) = }")
    return np.real(dI_dz)
```

Output:



Fixing bugs in the TIE(z, I, ϕ)

$$TIE(z, I, \phi) = \frac{-1}{k_0}\left(\nabla I \nabla\phi + I\nabla^2\phi\right)$$

$$\nabla I \nabla\phi = \mathcal{F}^{-1}\{(ik)\mathcal{F}\{I\}\}\mathcal{F}^{-1}\{(ik)\mathcal{F}\{\phi\}\}$$

$$I\nabla^2\phi = I\mathcal{F}^{-1}\{(ik)^2\mathcal{F}\{\phi\}\}$$

$$\therefore TIE(z, I, \phi) = \frac{-1}{k_0}\left(\mathcal{F}^{-1}\{(ik)\mathcal{F}\{I\}\}\mathcal{F}^{-1}\{(ik)\mathcal{F}\{\phi\}\} + I\mathcal{F}^{-1}\{(ik)^2\mathcal{F}\{\phi\}\}\right)$$

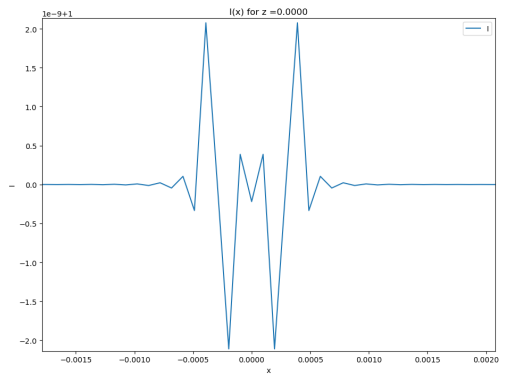
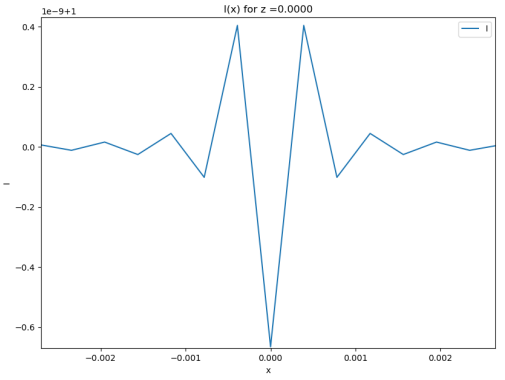
Plotting the intensity

To test if my code is correct I need to use a single value of z where I know interesting stuff happens…
I know that at the centre of my cylinder there is phase-shift. The centre of my cylinder is set at the z_c value
What index is that?
I ran my code and saved a list of state vectors one for each z value in the range of the while loop

```
67      # Propagation & loop parameters
68      i = 0
69      z_max = 100 * mm
70      z = -z_max
71      z_final = z_max
72      delta_z = 0.01 * mm # (n_z = 20000)
```

Since there are 20000 steps in the range given this delta_z. I approximated the z_c index as i = 10000.
After some trial and error I found the index where I(x, z) is something is i = 9363.
Any i higher than this yields an empty plot. Obviously this is also part of my bugs.

```
103     z = z_c
104     # dI_dz, dΦ_dz = dΨ_dz(z, Ψ)
105
106     psi_list = np.load("TIE/psi_list.npy")
107
108     I, Φ = psi_list[9363]
109
110     # # I Test plot
111     plt.plot(x, I, label="I")
112     # plt.xlim(-20 * mm, 20 * mm)
113     plt.xlabel("x")
114     plt.ylabel("I")
115     plt.legend()
116     plt.title(f"I(x) for {z =:.4f}")
117     plt.show()
```

	This plot looks super weird. The zig zags probably denote instability of the algorithm. I will try to fix this by modulating the relative sizes of delta_z and delta_x. If I increase the size delta_x while leaving delta_z untouched? from: x = np.linspace(-x_max, x_max, 2048, endpoint=False) to: x = np.linspace(-x_max, x_max, 512, endpoint=False)
	The intensity oscillation occurs less frequently. As expected. But the issue requires more work. I should try to use a different kind of IC for ϕ, then investigate what dI_dz looks like for that ϕ (and for the same all-ones I(x) I'm using now)

Testing I(x,z) and dI_dz(x,z) with a different ϕ

First I returned to 2048 as the number of x-steps

```
    # x-array parameters
    x_max = 100 * mm
    x = np.linspace(-x_max, x_max, 2048, endpoint=False)
    n = x.size
    delta_x = x[1] - x[0]
```

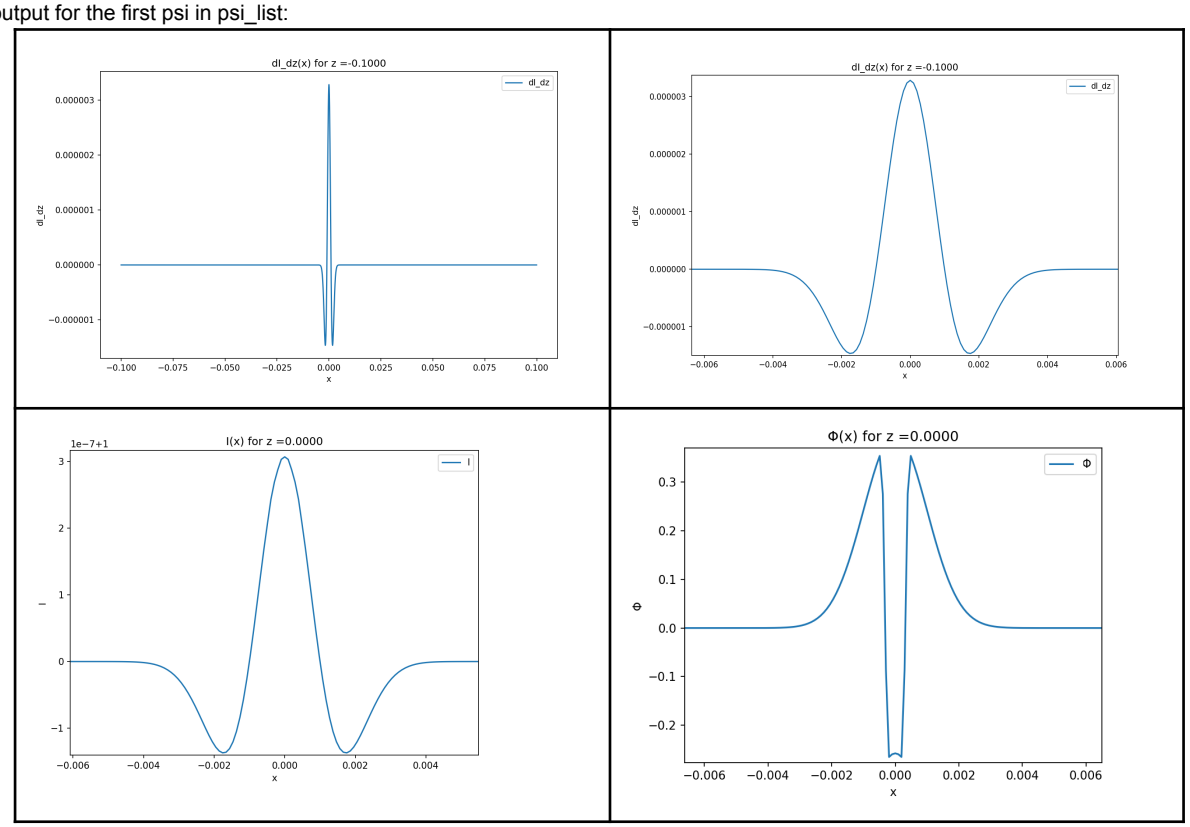
I defined a gaussian function to use as IC

```
def gaussian(E, amplitude=1 * mm, centre=0, sigma=1 * mm):
    return amplitude * (1 / (sigma * (np.sqrt(2 * np.pi)))) * (np.exp((-1 / 2) * (((E - centre) / sigma)**2)))
```

I'm still using I as an array of ones.

```

    # ICs
    I = np.ones_like(x)
    # Φ = np.zeros_like(x)
    Φ = gaussian(x) #testing

    Ψ = np.array([I, Φ])
```

dI_dz plot output for the first psi in psi_list:



Both I(x,z) and $\phi(x, z)$ $(gaussian)$hold a sensible relationship

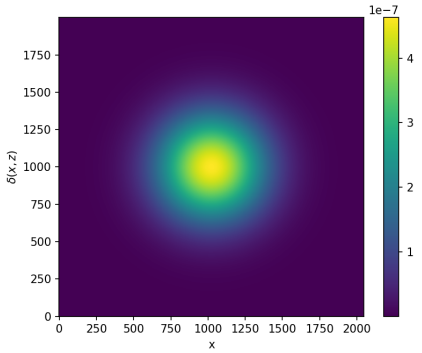
Trying out a smoother $\delta(x, z)$

Since I know that putting in a smooth ϕ seems to result in good-seeming behaviour, maybe a smooth δ(x,z) would help. I want to try a 2D Gaussian for δ instead of a 2D circular tophat.

Again it's just a test - I want to know what factors make a difference. If a smooth delta helps, then if after I change back to a sharp one the process breaks, then the problem must lie in between! That would mean a certain level of smoothness is required.

```

def gaussian2D(x, z, amplitude=1, x0=0, z0=0, sigma=1):
    return (
        amplitude
        * (np.exp((-1 / 2) * (((x - x0) ** 2 + (z - z0) ** 2) / (sigma ** 2))))
    )


def δ(x, z):
    # refractive index: constant inside the cylinder but zero everywhere else
    δ0 = 462.8 * nm
    # δ_array = np.zeros_like(x * z)
    # δ_array[(z - z_c) ** 2 + (x - x_c) ** 2 <= R ** 2] = δ0
    δ_array = gaussian2D(x, z, δ0, x_c, z_c, 25 * mm)
    return δ_array
```

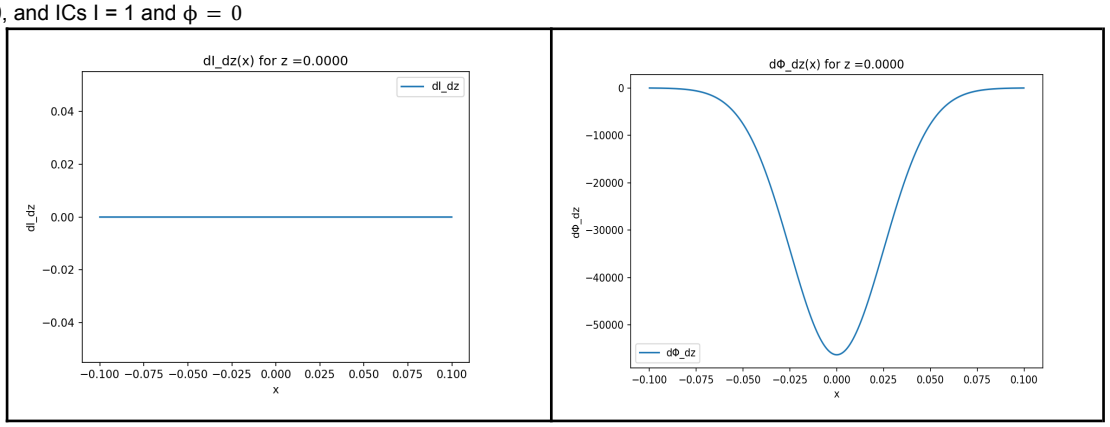
```
    # ###########
    ## PLAYING AROUND with δ ###
    z = np.linspace(-x_max, x_max, 2000, endpoint=False).reshape((2000, 1))
    δ_array = δ(x, z)
    plt.imshow(δ_array, origin='lower')
    plt.colorbar()
    plt.xlabel("x")
    plt.ylabel(r"$\delta(x, z)$")
    plt.show()
    ###########
```



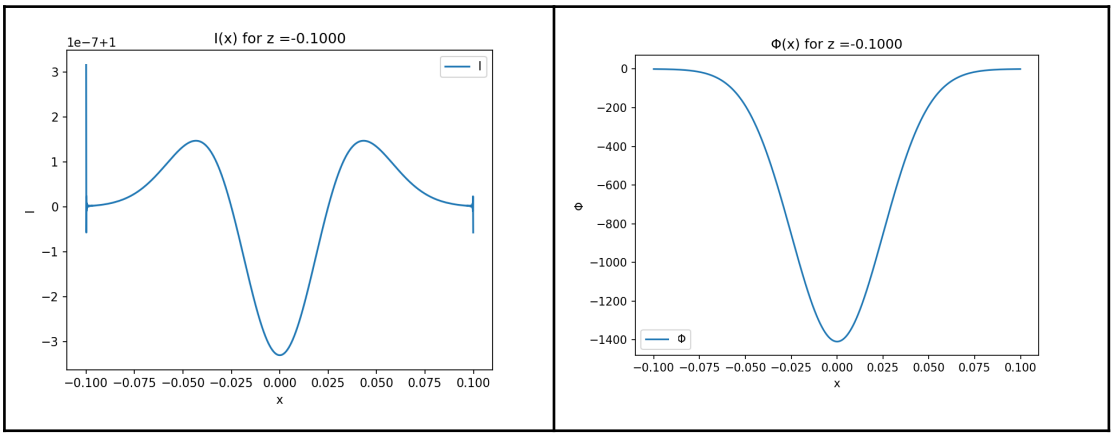
Blurry cylinder cross section

Using the smooth δ

For z = z_c = 0, and ICs I = 1 and $\phi = 0$



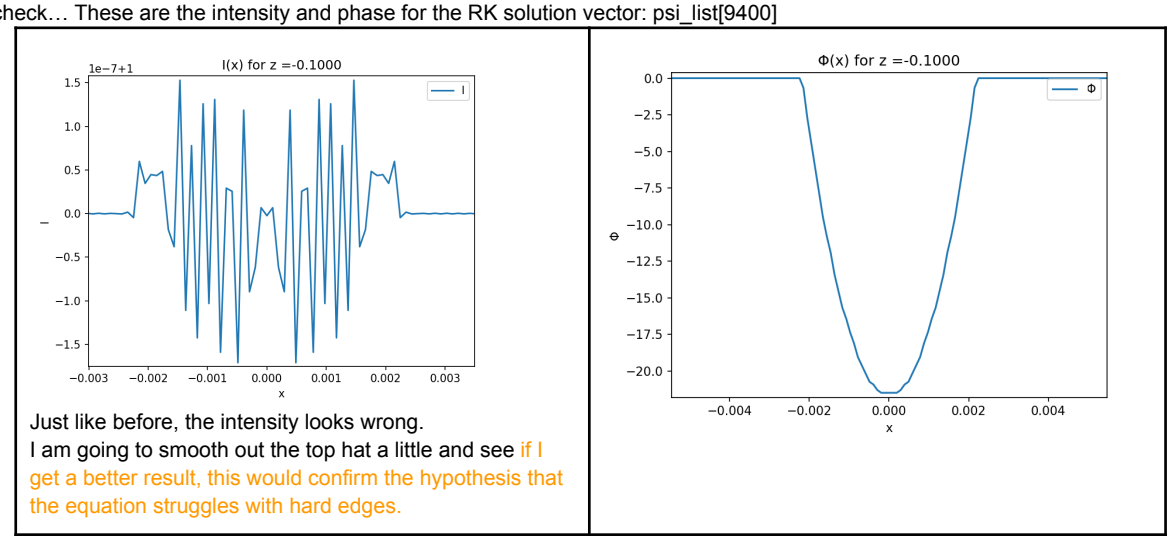
Testing the gaussian δ in RK



The intensity plot looks pretty good. Intensity moves in the direction of the phase gradient
The blow up at the edges is probably because my gaussian didn't go to zero fast enough.
From these plots I can see that my functions are working correctly.
The issues I was having might be occurring due to the equations being sensitive to the sharp edges of the top-hat δ

Trying out the top-hat δ

As a sanity check… These are the intensity and phase for the RK solution vector: psi_list[9400]



Just like before, the intensity looks wrong.
I am going to smooth out the top hat a little and see if I get a better result, this would confirm the hypothesis that the equation struggles with hard edges.

Week 5 (23/08/21 –29/08/21)

Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Meetings on **Monday 23/08** and **Wednesday 25/08**
    a. ~~Group: **2 pm (Didn't attend due to double booking with psychiatrist)**~~
    b. ~~one-on-one: **11 am**~~
2. **Read X-Ray Handbook**
3. **Read PyQt GUI book**
4. ~~**USE PROJ. APPROX.**~~

23/08/21

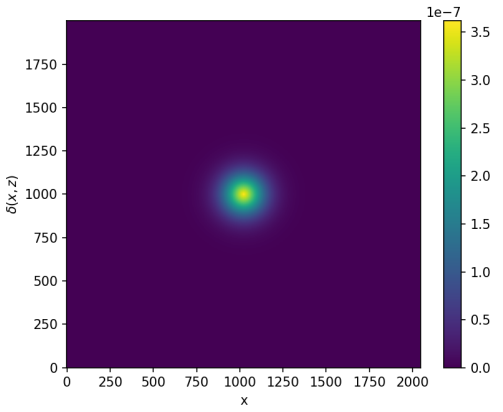
Settling on the softer top-hat

I want to use a sigmoid function to model a softer refractive index. The functional expression of the form:

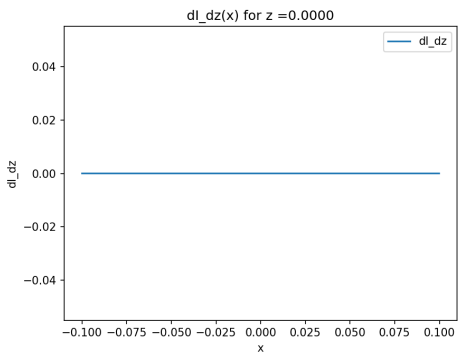
$$\delta(x,z) = \delta_0 \frac{1}{1+e^{(r-R))/\sigma}}$$

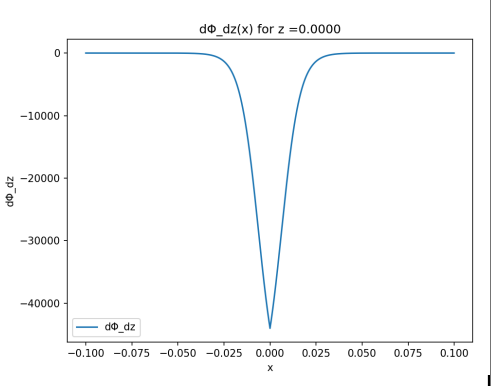
Where σ is the edge width parameter (the steepness of the curve). I chose σ to be smaller than the radius of the tophat δ) and $r(x,z) = \sqrt{x^2+z^2}$, the refractive index value reported in (Beltran et al. 2010) is $\delta 0$ = 462.8 nm.

```
def δ(x, z):
    # refractive index: constant inside the cylinder but zero everywhere else
    δ0 = 462.8 * nm
    # δ_array = np.zeros_like(x * z)
    # δ_array[(z - z_c) ** 2 + (x - x_c) ** 2 <= R ** 2] = δ0

    # refractive index: δ0 within the cylinder decreasing to zero at the edges.
    # Sigmoid function inspired:
    r = np.sqrt(x**2 + z**2)
    σ = 0.05 * mm
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ)))
    return δ_array
```



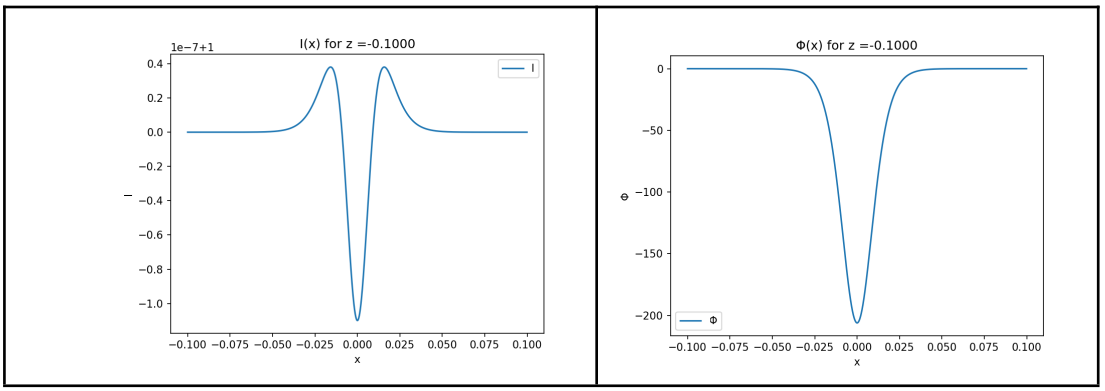

```
z = np.linspace(-x_max, x_max, 2000,
endpoint=False).reshape((2000, 1))
        # z = z_c
        # dI_dz, dΦ_dz = dΨ_dz(z, Ψ)
```



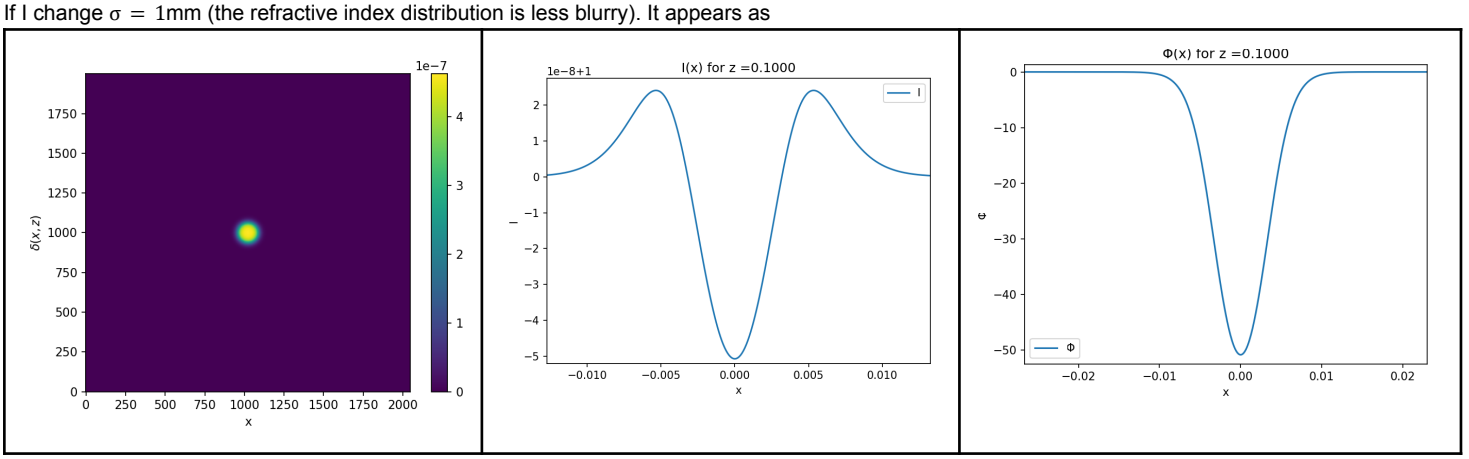


Testing Runge Kutta algorithm near the centre

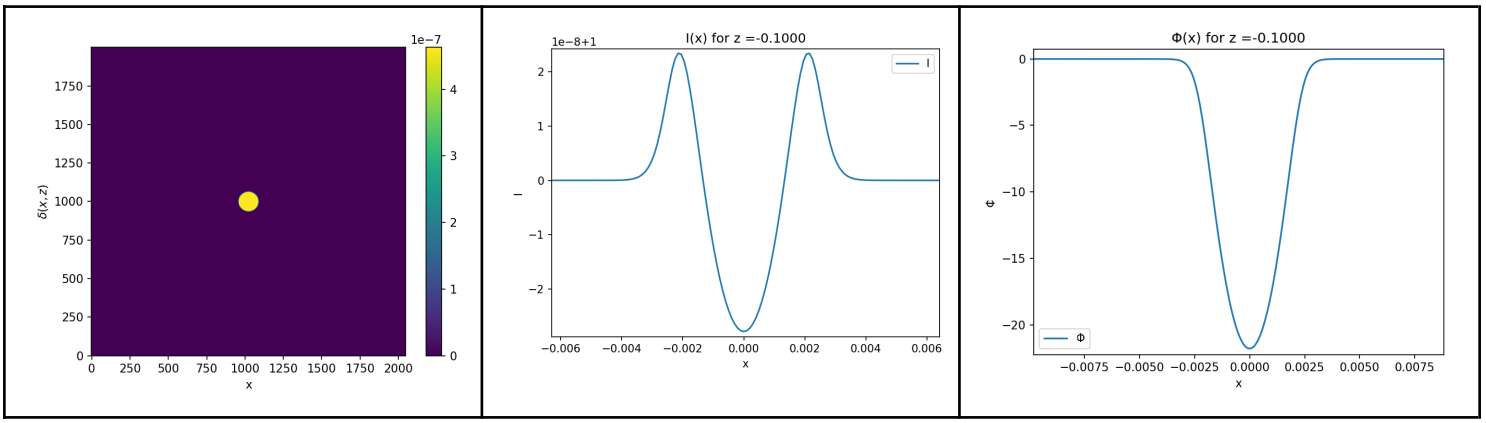
```
psi_list = np.load("TIE/psi_list.npy")
I, Φ = psi_list[9400]
```



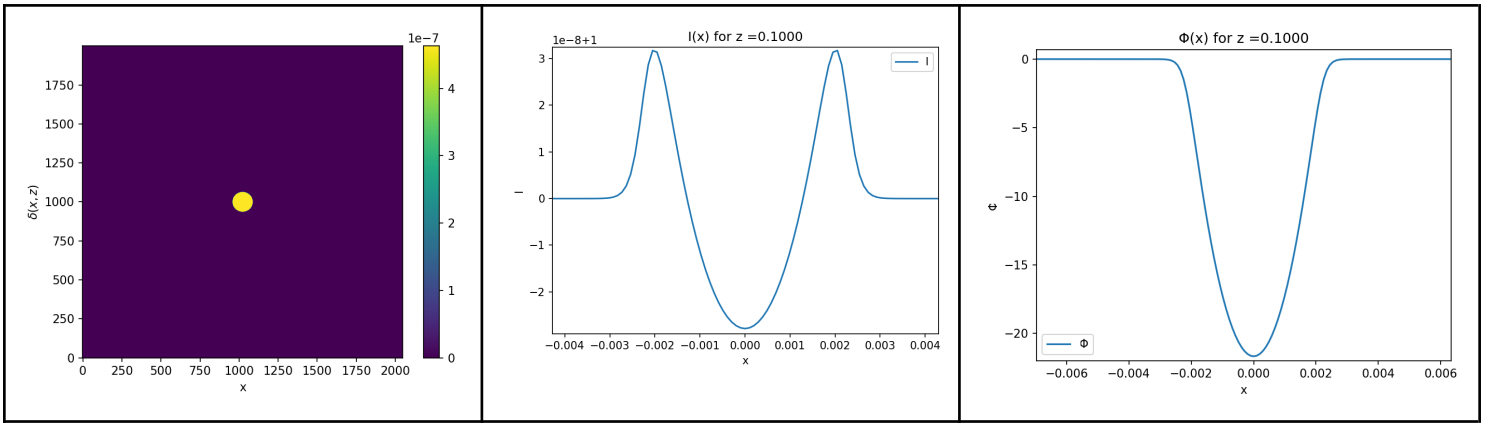
If I change $\sigma = 1$mm (the refractive index distribution is less blurry). It appears as



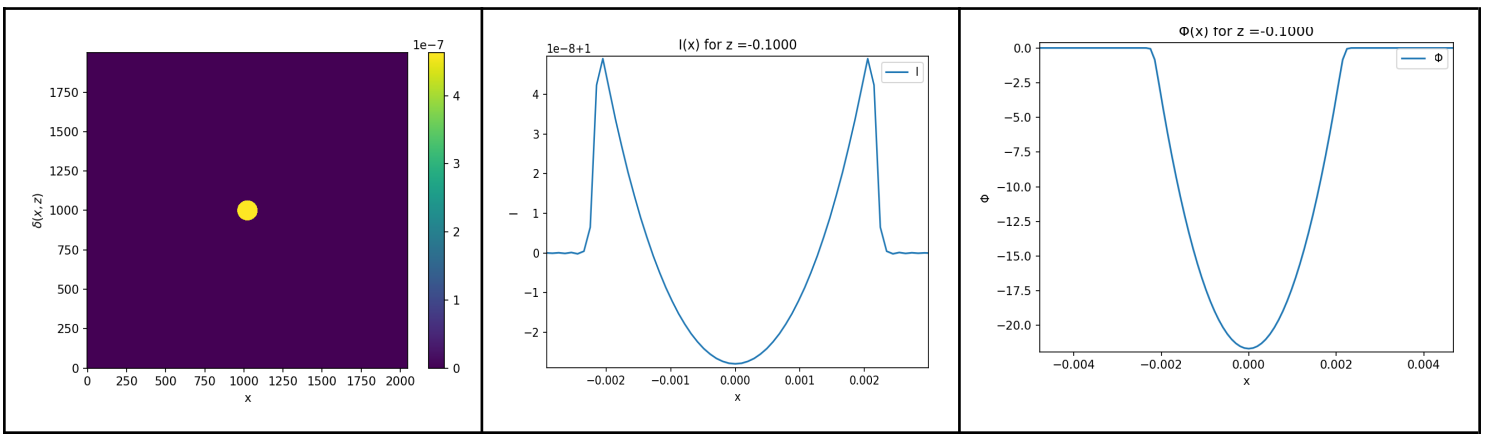
$\sigma = 0.1$mm



$\sigma = 0.05$mm

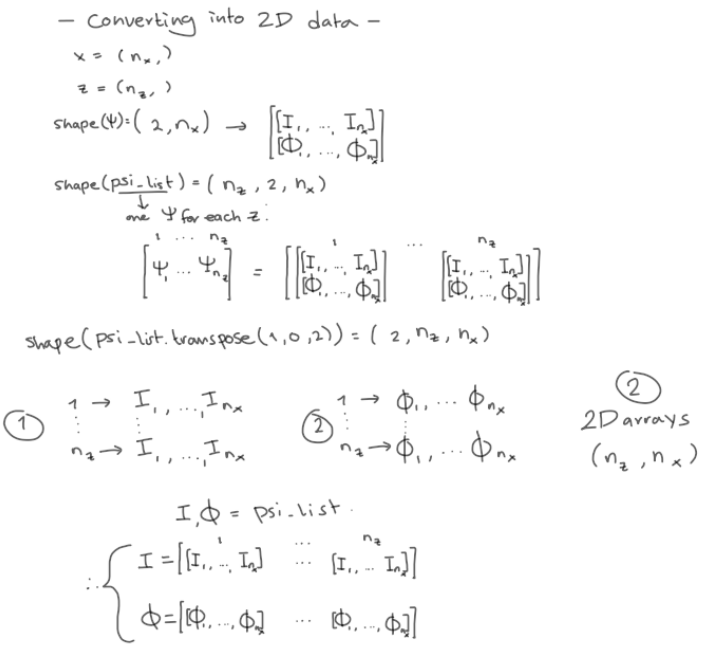


$\sigma = 0.01$mm



As I make the edge of the refractive index function more closely resemble a sharp edge the more unstable/sharp-wiggly the plot of the intensity becomes. Probably the numerics cannot do much better than this so I will continue using a **soft edge cylinder with** $\sigma = 0.05$mm.

24/08/21
In order to plot my data as an HSV imshow plot. I need to convert my data into 2D arrays.




```python
def complex_array_to_rgb(psi_list, i):
    '''Takes an array of complex numbers and converts it to an array of [r, g, b],
    where phase gives hue and saturaton/value are given by the absolute value.
    Especially for use with imshow for complex plots.'''

    I_list, Φ_list = psi_list
    # print(f"{all(np.imag(Φ_list) == 0)}")

    absmax = np.abs(I_list).max()
    # print(f"\n{absmax = } ")
    hsv = np.zeros(I_list.shape + (3,), dtype='float')
    # print(f"\n{hsv = }")
    hsv[:, :, 0] = Φ_list / (2 * np.pi)
    hsv[:, :, 1] = 1
    hsv[:, :, 2] = np.clip(np.abs(I_list) / absmax, 0, 1)
    print(f"\n{hsv =  }")
    rgb = matplotlib.colors.hsv_to_rgb(hsv)
    # print(f"\n{rgb = }")
    return rgb
```

Output:

```
[ana:code]$ python3 solver.py
solver.py:72: ComplexWarning: Casting complex values to real discards the imaginary part
  hsv[:, :, 0] = Φ_list / (2 * np.pi)

hsv =  array([[[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]],

       [[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]],

       [[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]],

       ...,

       [[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]],

       [[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]],

       [[0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        ...,
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569],
        [0.        , 1.        , 0.99651569]]])
Killed
[ana:code]$
```

the process is killed because my psi_list is enormous and my computer has very little RAM

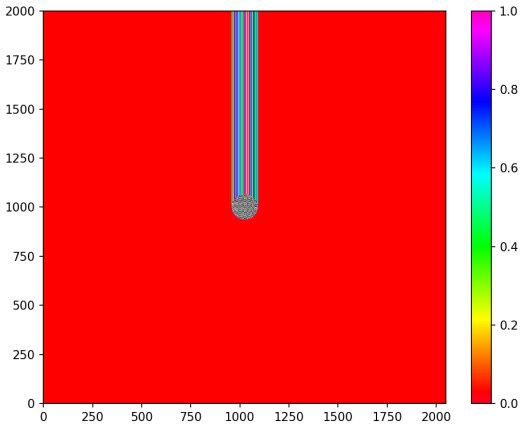
I checked and I require a high n_z value to have stability (my I plot becomes zig-zaggy and pointy with less z steps). Therefore I'm only saving every 10th state in psi_list

```
######################### RK LOOP ##############################

psi_list = []
while z < z_final:

    print(f"{i = }")

    # spatial evolution step
    Ψ = Runge_Kutta(z, delta_z, Ψ)
    # print(f"\n{Ψ = }")
    if not i % 10:
        psi_list.append(Ψ)
    i += 1
    z += delta_z

psi_list = np.array(psi_list)
print(f"{np.shape(psi_list)}")

np.save(f'psi_list.npy', psi_list)

######################### PLOTS & TESTS ##############################
```

Testing complex_array_to_rgb()

np.shape(psi_list) = (2, 2001, 2048)

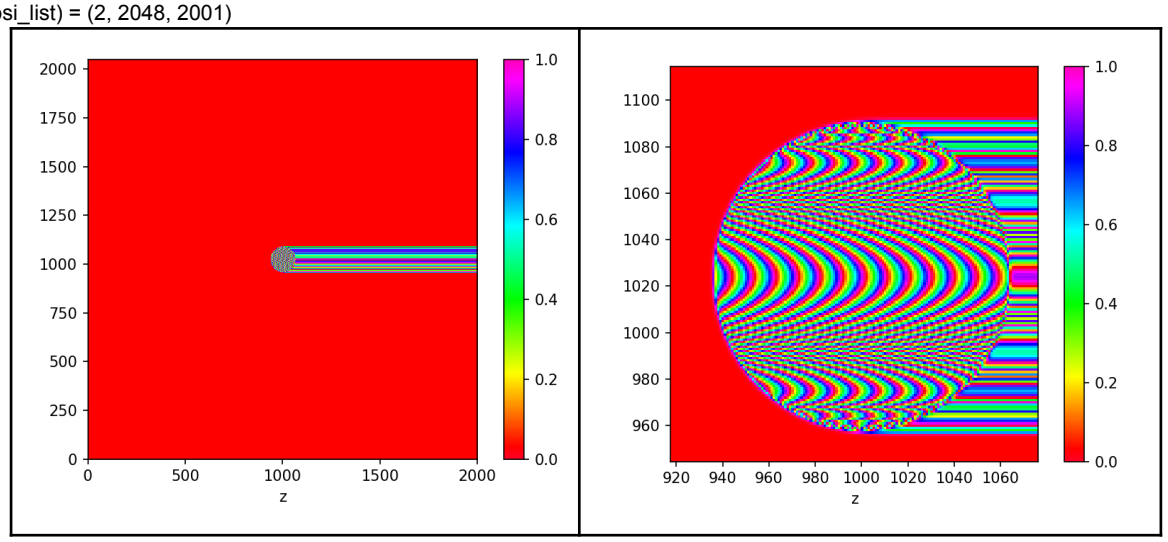
```
######################### PLOTS & TESTS ##############################
# Load and transpose
psi_list = np.load("psi_list.npy")
psi_list = psi_list.transpose(1, 0, 2)
# print(f"\n{np.shape(psi_list) = }")

# Simulation frame
plt.imshow(complex_array_to_rgb(psi_list),
           cmap="gist_rainbow",
           origin='lower')
plt.colorbar()
# plt.savefig(f'folder/{i:04d}.png')
plt.show()
# plt.clf()
```



I think my transposition is not quite right and that the horizontal axis should be z

```
137     ######################### PLOTS & TESTS ##############################
138     # Load and transpose
139     psi_list = np.load("psi_list.npy")
140     psi_list = psi_list.transpose(1, 2, 0)
141     print(f"\n{np.shape(psi_list) = }")
```

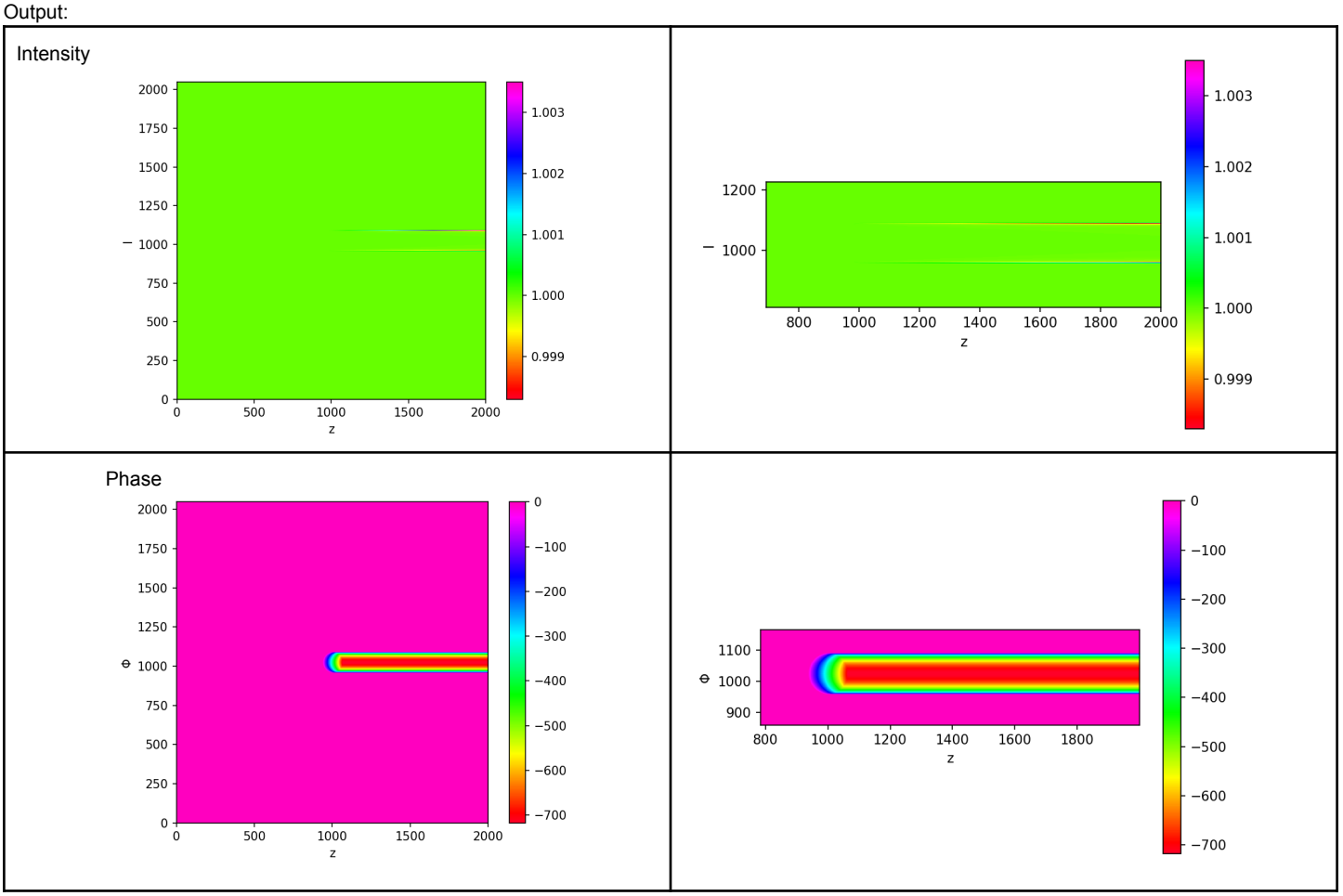
Output:
np.shape(psi_list) = (2, 2048, 2001)



Ok, I don't understand what is going on here…
Let's backtrack again
Plotting the I_list and ϕ_list separately

```
######################## PLOTS & TESTS ############################
# Load and transpose
psi_list = np.load("psi_list.npy")
psi_list = psi_list.transpose(1, 2, 0)
# print(f"\n{np.shape(psi_list) = }") # this returns np.shape(psi_list) = (2, 2048, 2001)

I_list, Φ_list = psi_list

############################

plt.imshow(I_list,
           cmap="gist_rainbow",
           origin='lower')
plt.colorbar()
plt.xlabel("z")
plt.ylabel("I")
plt.show()

############################


plt.imshow(Φ_list,
           cmap="gist_rainbow",
           origin='lower')
plt.colorbar()
plt.xlabel("z")
plt.ylabel("Φ")
plt.show()
```

Output:

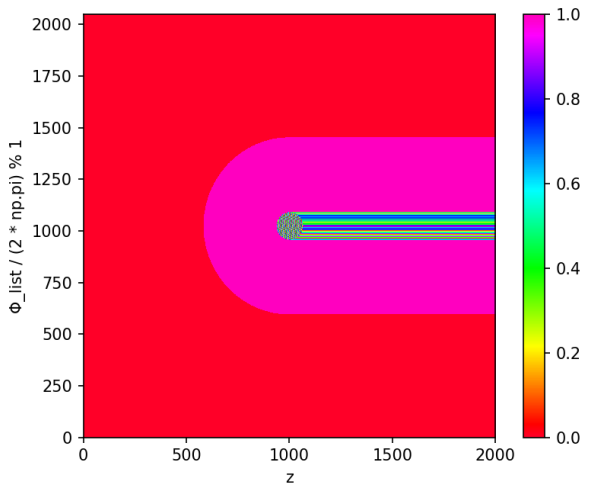
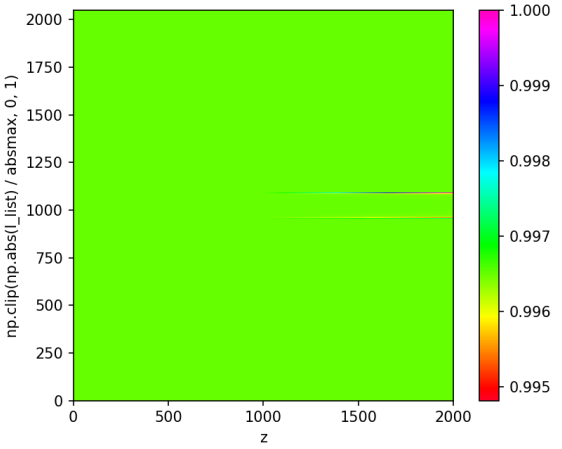
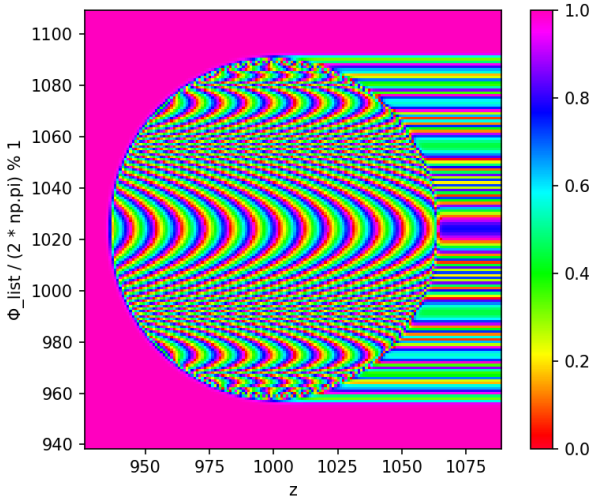


Yay I think that this looks like phase contrast!
Phase increasing at constant rate inside object ✅
Intensity showing wiggles at edge of object ✅
SO…. what is going on in my array2D_to_rgb() function (I changed the name of complex_array_to_rgb so the name made more sense)

```python
def array2D_to_rgb(psi_list):
    '''Takes a 2D array of numbers and converts it to an array of [r, g, b],
    where phase gives hue and saturaton/value are given by the absolute value.
    Especially for use with imshow plots.'''

    I_list, Φ_list = psi_list

    absmax = np.abs(I_list).max()
    hsv = np.zeros(I_list.shape + (3,), dtype='float')
    # hsv[:, :, 0] = Φ_list / (2 * np.pi) % 1

    plt.imshow(Φ_list / (2 * np.pi) % 1,
               cmap="gist_rainbow",
               origin='lower')
    plt.colorbar()
    plt.xlabel("z")
    plt.ylabel("Φ_list / (2 * np.pi) % 1")
    plt.show()

    # hsv[:, :, 1] = 1

    # hsv[:, :, 2] = np.clip(np.abs(I_list) / absmax, 0, 1)

    plt.imshow(np.clip(np.abs(I_list) / absmax, 0, 1),
           cmap="gist_rainbow",
           origin='lower')
    plt.colorbar()
    plt.xlabel("z")
    plt.ylabel("I")
    plt.show()

    # rgb = matplotlib.colors.hsv_to_rgb(hsv)
    # print(f"\n{rgb = }")
    # return rgb
```

```python
######################## PLOTS & TESTS ##############################
# Load and transpose
psi_list = np.load("psi_list.npy")
psi_list = psi_list.transpose(1, 2, 0)
# print(f"\n{np.shape(psi_list) = }") # this returns np.shape(psi_list) = (2, 2048, 2001)

array2D_to_rgb(psi_list)
```

Output:

Φ_list / (2 * np.pi) % 1	np.clip(np.abs(I_list) / absmax, 0, 1)
	
	I am not sure if this is what the phase should look like… I guess I thought it would look like wavefronts of constant phase. I asked Chris and he said that this is probably right. He said that unless I subtract 1.0 from the intensity and plot (I - I_0), we won't see much since I is so close to 1.0 that unfortunately the normalising code for I expects something with a single sign, whereas if I plot (I - I_0) I will have a result that is sometimes negative and sometimes positive. What you would want is to map the range (I_min, I_max) onto (0,1) there.

```python
    absmax = np.abs(I_list).max()
    absmin = np.abs(I_list).min()
    print(f"{absmax =}")
    print(f"{absmin =}")
```

```
[ana:code]$ python3 solver.py
absmax =1.0034964897604162
absmin =0.9982962502462929
```

Normalising I

* Mapping I_{max} to 1 and I_{min} to 0

let the transformations be

① $0 = mI_{min} + C$

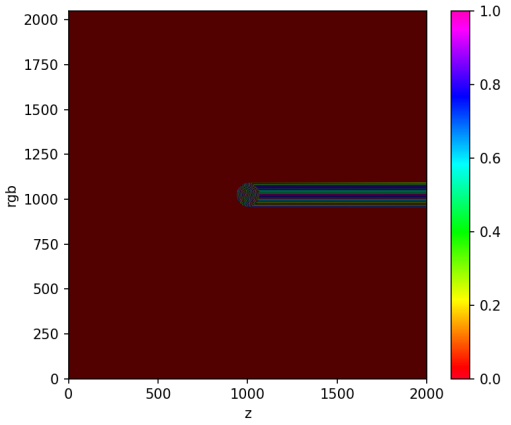
② $1 = mI_{max} + C$

re-write ①: $C = -mI_{min}$

substitute ① in ②: $1 = mI_{max} - mI_{min}$

$\therefore 1 = m(I_{max} - I_{min})$

$\therefore m = \frac{1}{(I_{max} - I_{min})}$



This visualisation doesn't seem very useful since now the average brightness is low. Given that the wrapped intensity is so localised and small and that the phase has so much going on.

25/08/21

MEETING DAY!

Questions for MK:

1. What do you think of my solution to apparent instability of the intensity?
   I smoothed out the top hat a little to test the hypothesis that the numerics struggle with hard edges.
   to model a softer refractive index. I used a functional expression of the form:

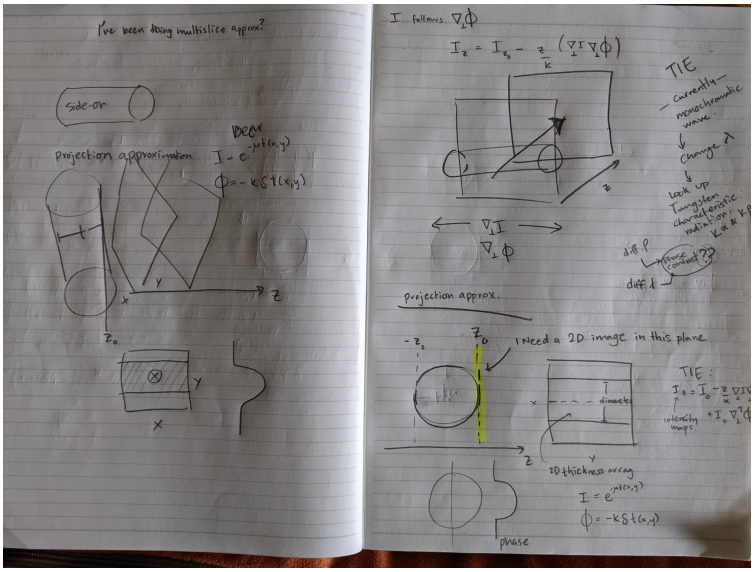
   $$\delta(x,z) = \delta_0 \frac{1}{1 + e^{(r-R))/\sigma}}$$

   where $\sigma = 0.05$mm and r = $\sqrt{x^2 + z^2}$
2. Does the colour map output match what would be expected from phase contrast?Am I seeing aliasing?
   - Should I increase resolution in x?
   - the Nyquist frequency. Which is a new term for me.
     delta_x must be small enough to be able to resolve the nyquist mode within the delta_z specified.
3. Should I visualise the spatial evolution in z (as an animation?)
4. Phase contrast imaging… How do I proceed?

Turns out that even though I have made good progress and my results appear to make sense, this is not the type of simulation that MK had in mind. It looks like he hasn't actually thought about things the way that I've been. He says that the idea behind the simulation he had in mind is actually using the ***projection approximation****.*
According to MK I've been doing something similar to the ***multislice approximation***

Here are some notes on that multislice approximation method:
This methodology, of which there are
many variants, was originally developed
in the context of electron diffraction by
Cowley and Moodie (1957). Our outline, of one such
variant, will make use of both the angular spectrum formalism
and the projection approximation. Using these tools, we will
construct a multislice algorithm for the scattering of coherent scalar X-rays from
a given potential, which is sufficiently weakly scattering for us to be able to
neglect backscatter.(Paganin 99)
The object is assumed to be sufficiently weakly scattering such that the wave-field is forward
propagating, over any of the sequence of parallel planes marked A through H. ...the scatterers are
described by their three-dimensional distribution of refractive index n (x, y, z),
with z 0 ≤ z ≤ z_0 + Δ (Paganin 100).

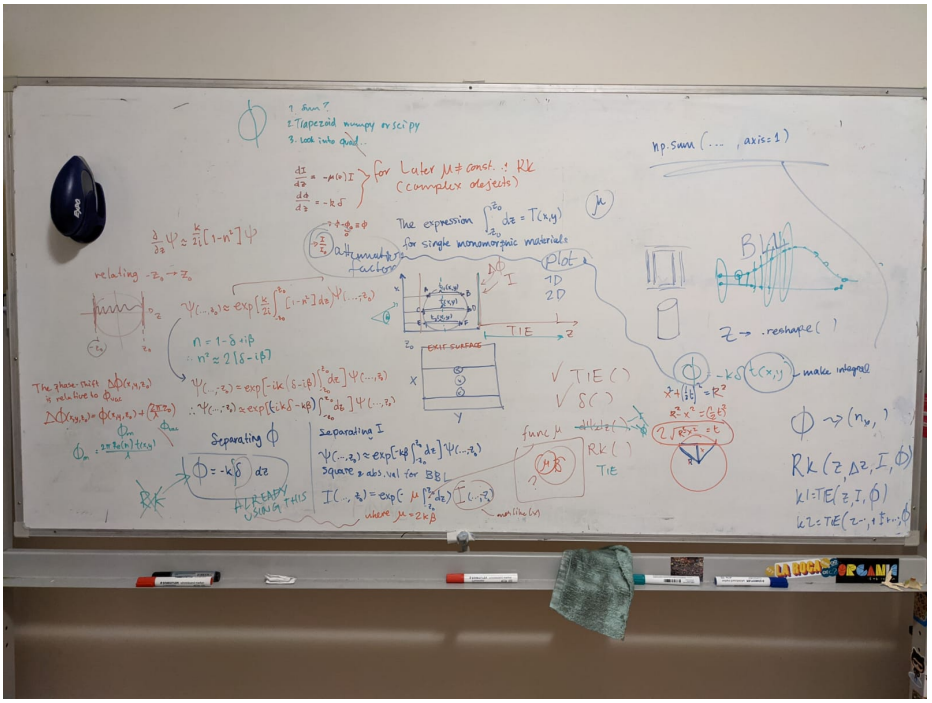
Using the projection approximation



assuming that the value of the wave-field at the exit-surface z = z 0 , is entirely determined by the phase and amplitude shifts that are accumulated along streamlines of the unscattered beam, and since the transverse Laplacian is the only portion of the inhomogeneous paraxial equation that couples neighbouring ray trajectories, neglecting this term amounts to the projection approximation (Paganin 73).

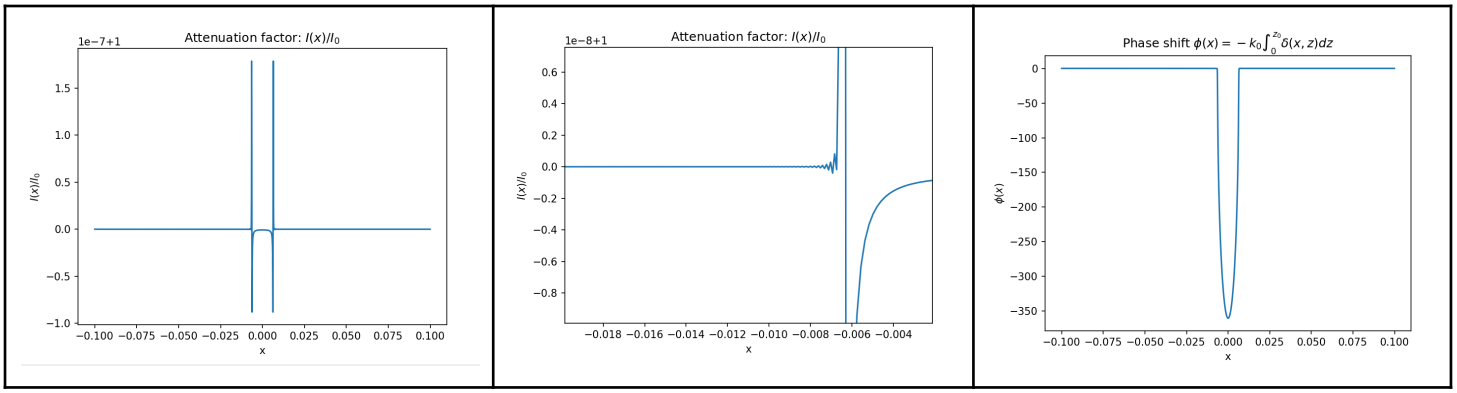
once the intensity and phase of the scattered wave-field have been estimated over the plane z = z 0 using the projection approximation, this information can subsequently be used to calculate the wave-field which results when this exit-surface field is propagated from z = z 0 to z = z 1 > z 0 (see Fig. 2.2). This propagation could be effected using an appropriate diffraction integral, including any of the means for propagating coherent wave-fields which were introduced in the previous chapter. Alternatively, one can compute the image produced by a well characterized imaging system which takes, as input, the exit field over the surface z = z 0 (cf. Chapter 4) (Paganin 77)

white-board-mind-vomit



27/08/21

My current code works. Nevertheless, I will adapt it to fit the projection approximation.
To implement the projection approximation I need to use the Beer-Lambert law and the integral form of the phase shift.



I have some ringing. My numerics are probably struggling with the σ parameter and with resolving the **nyquist frequency**.
I can fix this by increasing sigma to $\sigma = 0.1\,mm$.

Code changes

I noticed that I am not using my z_arrays in the δ and μ functions..I use the z step from the RK loop and so my plot of the attenuation factor is wrong

```python
def δ(x, z):
    '''Refractive index: δ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    δ0 = 462.8 * nm
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.1 * mm
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ)))
    return δ_array # np.shape(δ_array) = (n_x,)


def μ(x, z):
    '''attenuation coefficient: μ0 within the cylinde
    decreasing to zero at the edges Sigmoid inspired:'''
    μ0 = 41.2 # per meter
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.1 * mm
    μ_array = μ0 * (1 / (1 + np.exp((r - R) / σ)))
    return μ_array # np.shape(μ_array) = (n_x,)


def phase(x):
    # phase gain as a function of the cylinder refractive index
    z = np.linspace(-2 * R, 2 * R, 2000, endpoint=False).reshape((2000, 1))
    dz = z[1] - z[0]
    ϕ = np.sum(-k0 * δ(x, z) * dz, axis=0)
    return ϕ # np.shape(ϕ) = (n_x,)


def BLL(x):
    # Brute force integral to find the IC of the intensity (z = z_0)
    z = np.linspace(-2 * R, 2 * R, 2000, endpoint=False).reshape((2000, 1))
    dz = z[1] - z[0]
    I = np.exp(- np.sum(μ(x, z) * dz, axis=0)) * I_0
    return I # np.shape(I) = (n_x,)


def TIE(z, I, ϕ):
    '''The intensity and phase evolution of a paraxial monochromatic
    scalar electromagnetic wave on propagation'''
    dI_dz = (-1 / k0) * (
        np.real(
            ifft(1j * k * fft(I)) * ifft(1j * k * fft(ϕ))
            + I * ifft((1j * k) ** 2 * fft(ϕ))
        )
    )
    return dI_dz # np.shape(dI_dz) = (n_x,)


def Runge_Kutta(z, delta_z, I, ϕ):
    # spatial evolution 4th order RK
    # z is single value, delta_z is step
    k1 = TIE(z, I, ϕ)
    k2 = TIE(z + delta_z / 2, I + k1 * delta_z / 2, ϕ)
    k3 = TIE(z + delta_z / 2, I + k2 * delta_z / 2, ϕ)
    k4 = TIE(z + delta_z, I + k3 * delta_z, ϕ)
    return I + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)  # shape = (n_x,)

def globals():
    # x-array parameters
    x_max = 100 * mm
    x = np.linspace(-x_max, x_max, 2048, endpoint=False)
    delta_x = x[1] - x[0]
    n = x.size

    # X-ray beam parameters
    λ = 0.05166 * nm  # x-rays wavelength
    k0 = 2 * np.pi / λ  # x-rays wavenumber

    # Cylinder parameters
    D = 12.75 * mm
    R = D / 2
    z_c = 0 * mm
    x_c = 0 * mm

    # For Fourier space
    k = 2 * np.pi * np.fft.fftfreq(n, delta_x)


    return x, k0, R, z_c, x_c, k


# ----------------------------------------------------------------------------- #


if __name__ == '__main__':

    x, k0, R, z_c, x_c, k = globals()

    # RK Propagation loop parameters
    i = 0
    z = 0
    z_final = 100 * mm
    delta_z = 0.01 * mm  # (n_z = 10000)
```

Now my imaged object is centered correctly in (0,0)

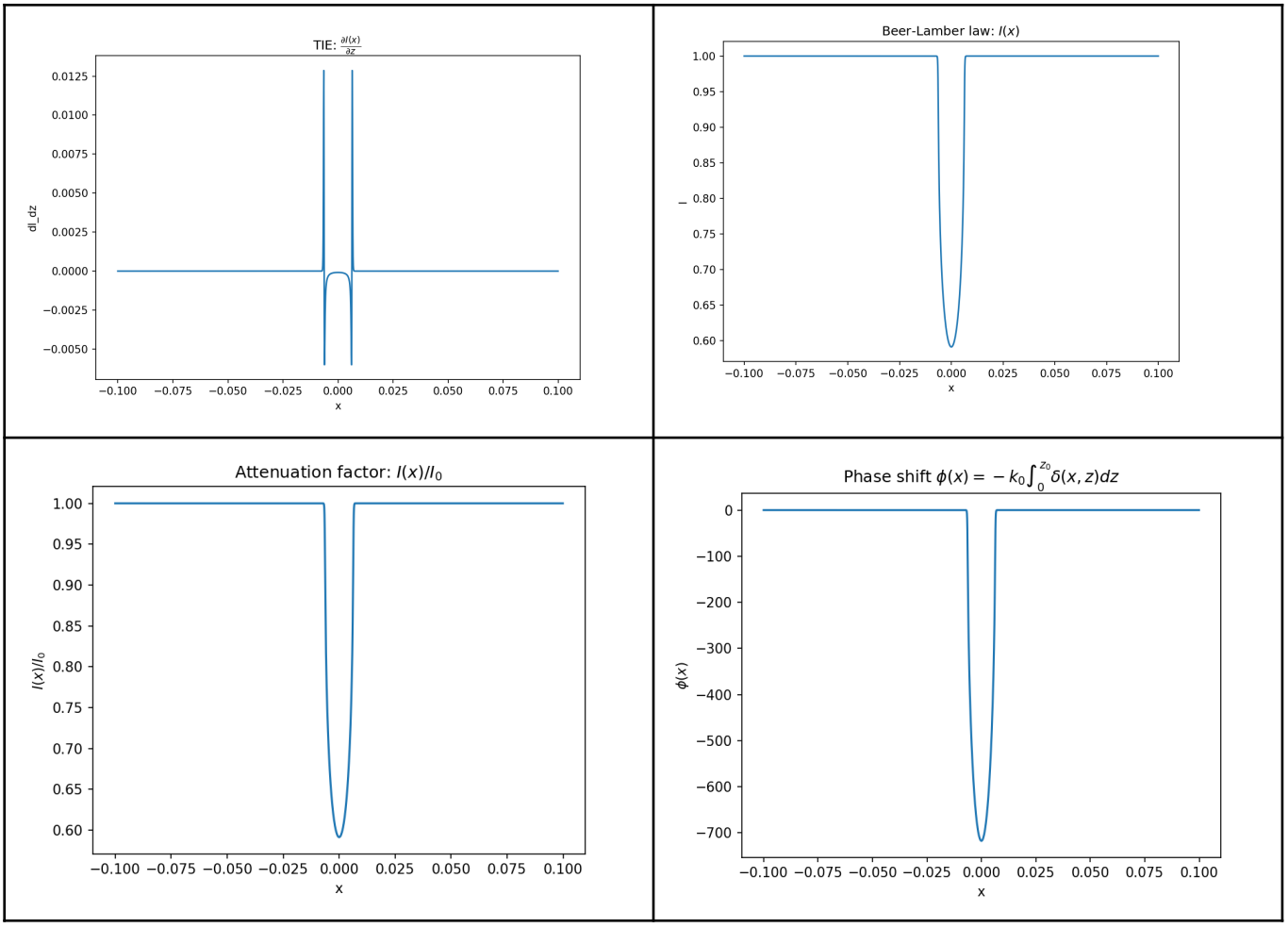
I clarified my thinking because I was confusing my z array with the parameter in the RK loop. The integration of ϕ and BLL is only for the object's thickness not for the propagation distance in the TIE.

This setup is cleaner

```
109     I_0 = np.ones_like(x)
110     ϕ = phase(x)
111
112     ######################### RK LOOP ##############################
113
114     I_list = []
115     while z < z_final:
116
117         print(f"{i = }")
118
119         # spatial evolution step
120         I = Runge_Kutta(z, delta_z, BLL(x), ϕ)
121         if not i % 10:
122             I_list.append(I)
123         i += 1
124         z += delta_z
125
126     I_list = np.array(I_list)
127     print(f"{np.shape(I_list) = }") #  np.shape(I_list) = (n_z / 10, n_x)
128
129
130     np.save(f'I_list.npy', I_list)

133      ######################## PLOTS & TESTS ############################
134      # Load file
135      I_list = np.load("I_list.npy") # np.shape(I_list) = (n_z / 10, n_x)
136
137      I = BLL(x)
138      dI_dz = TIE(z, I, ϕ)
139
140      # dI dz Test plot
141      plt.plot(x, dI dz)
142      plt.xlabel("x")
143      plt.ylabel("dI dz")
144      plt.title(r"TIE: $\frac{\partial I(x)}{\partial z}$ ")
145      plt.show()
146
147      # I Test plot
148      plt.plot(x, I)
149      plt.xlabel("x")
150      plt.ylabel("I")
151      plt.title(r"Beer-Lamber law: $I(x)$ ")
152      plt.show()
153
154      # PLOT ATTENUATION FACTOR I/I0 vs x after RK
155      plt.plot(x, I list[1000] / I_0)
156      plt.xlabel("x")
157      plt.ylabel(r"$I(x)/I_{0}$")
158      plt.title(r"Attenuation factor: $I(x)/I_{0}$ ")
159      plt.show()
160
161      # PLOT ϕ vs x
162      plt.plot(x, ϕ)
163      plt.xlabel("x")
164      plt.ylabel(r"$\phi(x)$")
165      plt.title(r"Phase shift $\phi(x) = -k_{0} \int^{z_{0}}_{0} \delta(x, z) dz$ ")
166      plt.show()
```

loop is slower because I am using the correct z thickness.

Output:



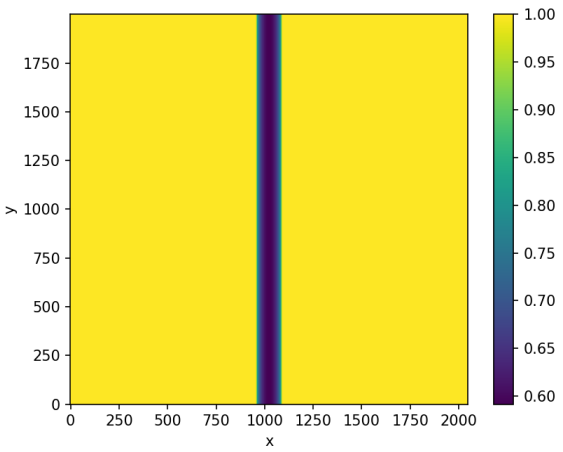
wow what! These plots look different from what I expected!?

I was confused!! I mislabelled the derivative of the intensity (TIE) as the attenuation factor plot. Chris said that my plot looks good now. We talked a little bit about my confusion and in hindsight it is very clear that the intensity should decrease as the X-ray propagates through the object. The TIE has nothing to do with the light propagation through the object but instead the projection approximation uses it to propagate the beam after the plane at z_0.

This is what I get for combining code approaches without **stopping to think about the physics.**
In fact the attenuation factor plot is equal to the BLL plot since $I_0 = 1$by my definition.

Making a 2D plot

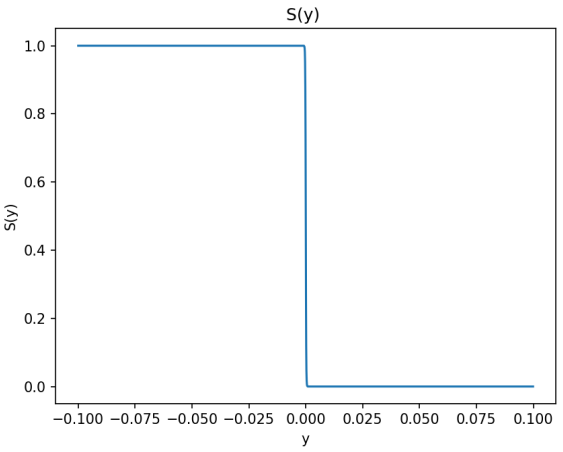
```
    # # TODO:
    # # PLOTS ATTENUATION FACTOR I/I0 vs x, z (2D)
    n_y = 2000
    I_list = np.load("I_list.npy") # np.shape(I_list) = (n_z / 10, n_x)
    I = I_list[1000]
    I_2D = np.broadcast_to(I, (n_y, n_x))

    y = np.linspace(-100 * mm, 100 * mm, n_y, endpoint=False)
    plt.imshow(I_2D / I_0, origin='lower')
    plt.colorbar()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```



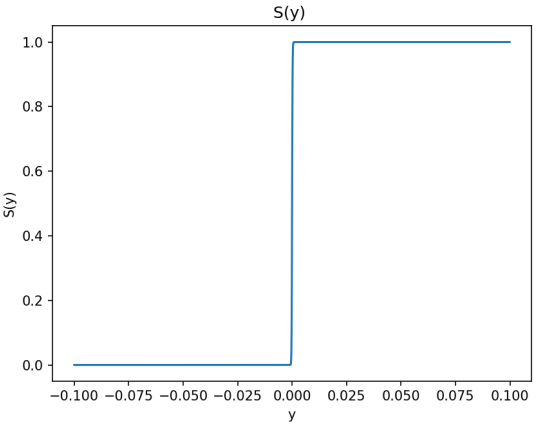
29/08/21

I need to define my y dimension to create a true 2D plot

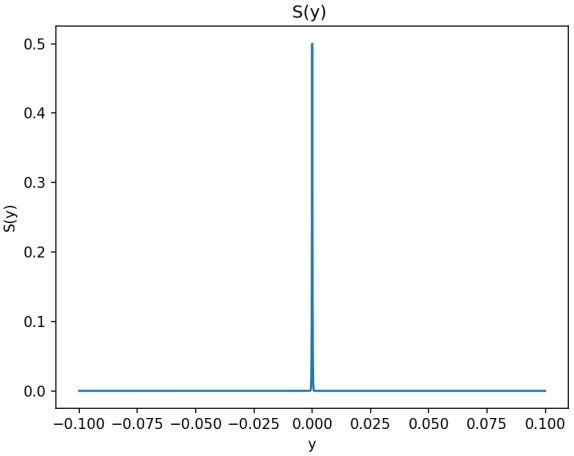
```
def y_sigmoid(y):
    σ = 0.1 * mm
    S = (1 / (1 + np.exp((y - y_c) / σ)))
    return S # np.shape = (n_y,)

if __name__ == '__main__':

    # y-array parameters
    n_y = 2000
    y_max = 100 * mm
    y = np.linspace(-y_max, y_max, n_y, endpoint=False)
    delta_y = y[1] - y[0]
    n = y.size

    # Cylinder parameters
    y_c = 0 * mm

    # dI_dz Test plot
    plt.plot(y, y_sigmoid(y))
    plt.xlabel("y")
    plt.ylabel("S(y)")
    plt.title("S(y) ")
    plt.show()
```



S = (1 / (1 + np.exp(-(y - y_c) / σ)))



S = (1 / (1 + np.exp(np.abs(-(y - y_c)) / σ)))



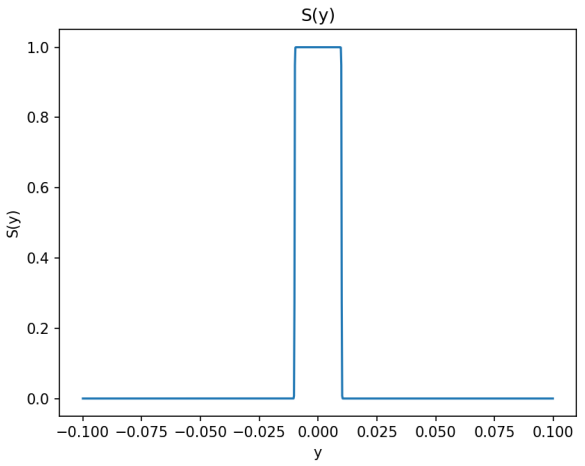
Ok, so this needs some thickness!

Let **h = 20 mm** be the height of my cylinder.

According to http://roperld.com/science/DoubleSigmoid.pdf:
A simple and computationally adequate formulation for the **double sigmoid** is (Roper 2)

$$S(y) = \frac{1}{2}\left[\tanh\left(\frac{y - h/2}{\sigma}\right) - \tanh\left(\frac{y + h/2}{\sigma}\right)\right]$$

```python
def y_sigmoid(y):
    σ = 0.1 * mm
    S = 1 / 2 * (np.tanh(-(y - h/2)/ σ) - np.tanh(-(y + h/2)/ σ))
    return S # np.shape = (n_y, 1)
```




```python
def δ(x, y, z):
    '''Refractive index: δ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    δ0 = 462.8 * nm
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.1 * mm
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
    return δ_array # np.shape(δ_array) = (n_y, n_x)


def μ(x, y, z):
    '''attenuation coefficient: μ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    μ0 = 41.2 # per meter
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.1 * mm
    μ_array = μ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
    return μ_array # np.shape(μ_array) = (n_y, n_x)


def phase(x, y):
    # phase gain as a function of the cylinder refractive index
    z = np.linspace(-2 * R, 2 * R, 2000, endpoint=False).reshape((2000, 1, 1))
    dz = z[1] - z[0]
    Φ = np.sum(-k0 * δ(x, y, z) * dz, axis=0)
    print(f"\n\n{np.shape(Φ) = }")
    return Φ # np.shape(Φ) = (n_y, n_x)


def BLL(x, y):
    # Brute force integral to find the IC of the intensity (z = z_0)
    z = np.linspace(-2 * R, 2 * R, 2000, endpoint=False).reshape((2000, 1, 1))
    dz = z[1] - z[0]
    I = np.exp(- np.sum(μ(x, y, z) * dz, axis=0)) * I_0
    print(f"\n\n{np.shape(I) = }")
    return I # np.shape(I) = (n_y, n_x)
```

```
ana@ana-XPS-13-9343: ~/Desktop/proj2/X-ray-kitchen/code
proj_approx_2D.py:16: RuntimeWarning: overflow encountered in exp
  S = 1 / (1 + np.exp(np.abs(-(y - y_c)) / σ))


np.shape(Φ) = (2048,)
proj_approx_2D.py:36: RuntimeWarning: overflow encountered in exp
  μ_array = μ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)


np.shape(I) = (2000, 2048)
[ana:code]$ python3 proj_approx_2D.py
proj_approx_2D.py:26: RuntimeWarning: overflow encountered in exp
  δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
proj_approx_2D.py:16: RuntimeWarning: overflow encountered in exp
  S = 1 / (1 + np.exp(np.abs(-(y - y_c)) / σ))
Traceback (most recent call last):
  File "proj_approx_2D.py", line 132, in <module>
    Φ = phase(x, y)
  File "proj_approx_2D.py", line 44, in phase
    Φ = np.sum(-k0 * δ(x, y, z) * dz, axis=0)
  File "proj_approx_2D.py", line 26, in δ
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
MemoryError: Unable to allocate array with shape (2000, 2000, 2048) and data type float64
[ana:code]$
```

This error is because of my puny computer!!!
Also, I just realised that my BLL() function is written wrong! :(

MemoryError
I cannot construct a 3D array of size 2000 * 2000 * 2048 because my RAM is only 8GB.
Although I can construct each slice of the z-integral individually and get them in an array of the correct shape (n_y, n_x) this doesn't require any broadcasting into a 3D array.

```python
def phase(x, y):
    # phase gain as a function of the cylinder refractive index
    z = np.linspace(-2 * R, 2 * R, 2000, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    Φ = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        Φ += -k0 * δ(x, y, z_value) * dz
    print(f"\n\n{np.shape(Φ) = }")
    return Φ # np.shape(Φ) = (n_y, n_x)
```

```
0.002460750000000001
0.0024735000000000017
0.0024862500000000006
0.0024990000000000012
0.002511750000000002
0.0025245000000000007
0.0025372500000000013
0.00255
0.002562750000000001
0.0025755000000000014
0.0025882500000000003
```

Similarly for the Beer-Lambert law

```python
def BLL(x, y):
    # TIE IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
    z = np.linspace(-2 * R, 2 * R, 1026, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    F = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        F += - μ(x, y, z_value) * dz
    I = np.exp(- F) * I_0
    return I # np.shape(I) = (n_y, n_x)
```

The resultant arrays are shaped (n_x, n_y)

Modifying the TIE() requires that I change k $\rightarrow (k_x + k_y)$

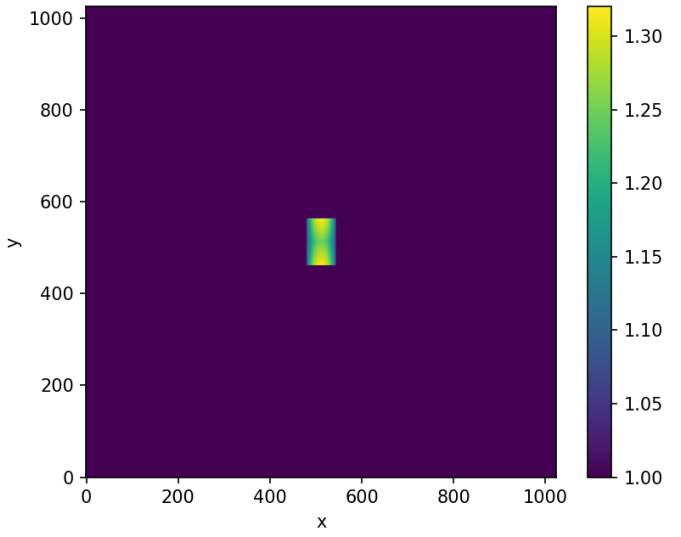
```python
    # For Fourier space
    kx = 2 * np.pi * np.fft.fftfreq(size_x, delta_x)
    ky = 2 * np.pi * np.fft.fftfreq(size_y, delta_y).reshape(size_y, 1)
    k = (kx + ky) # np.shape(k) = (size_x, size_y)
```

```
ana@ana-XPS-13-9343: ~/Desktop/proj2/X-ray...
i = 983
i = 984
i = 985
i = 986
i = 987
i = 988
i = 989
i = 990
i = 991
i = 992
i = 993
i = 994
i = 995
i = 996
i = 997
i = 998
i = 999
np.shape(I_list) = (100, 1025, 1024)

real    10m37.276s
user    9m51.393s
sys     0m44.936s
[ana:code]$
```

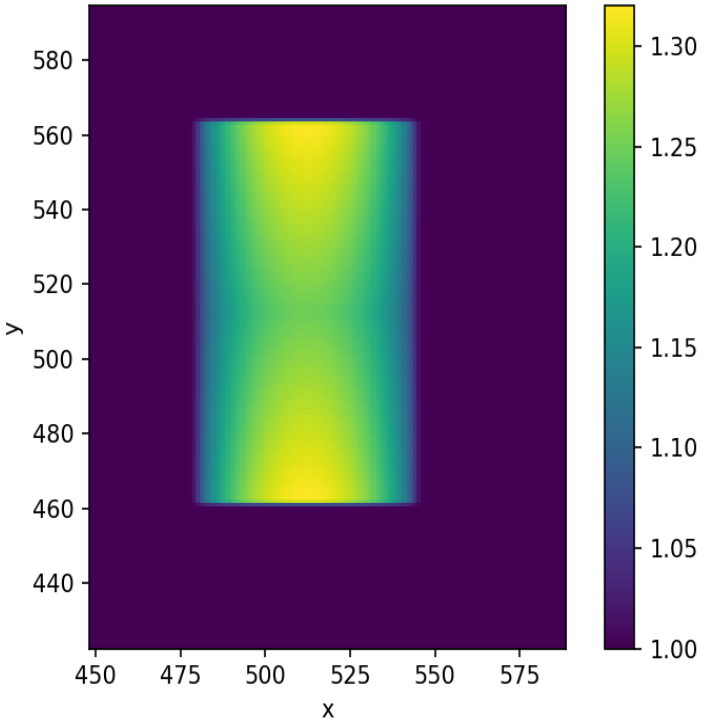
After attempting to run my code I realised that the time spent is too long so by decreasing the number of points in my RK loop I will be able to check sooner if my code is working correctly. I changed delta_z = 0.1 * mm therefore my number of z steps is n_z = 1000.
(I am only saving 1/10 of the I arrays).

My code is still quite slow.

```
# # PLOTS ATTENUATION FACTOR I/I0 vs x, y (2D)
# I_list = np.load("I_list.npy") # np.shape(I_list) = (100, 1025, 1024)

I = I_list[99,:, :]
# print(f"{np.shape(I) = }")

plt.imshow(I / I_0, origin='lower')
plt.colorbar()
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

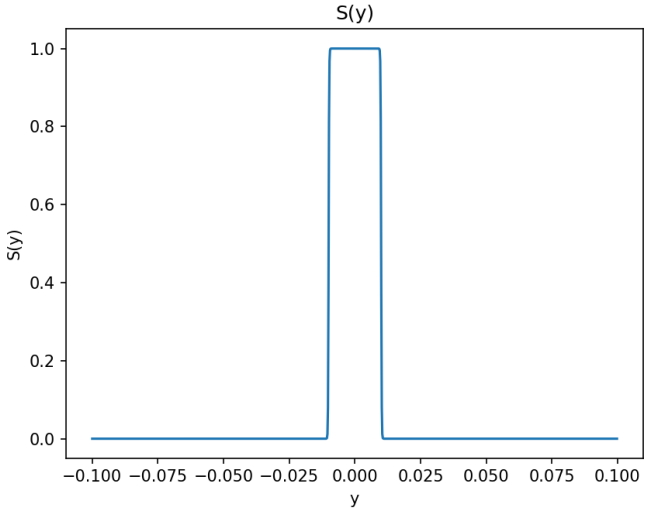


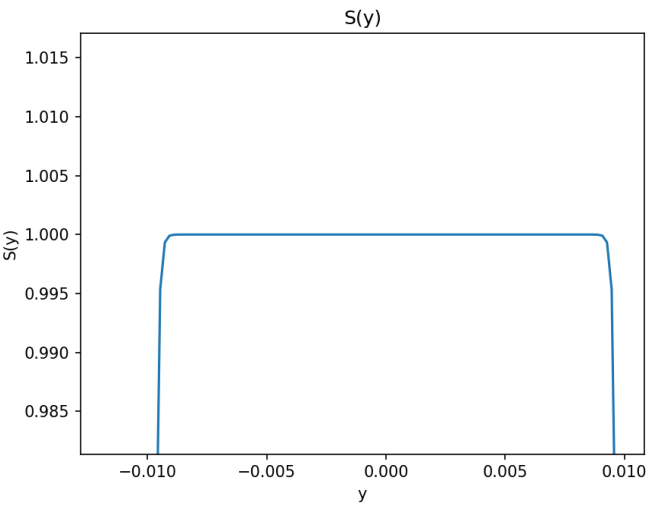
ENHANCE!

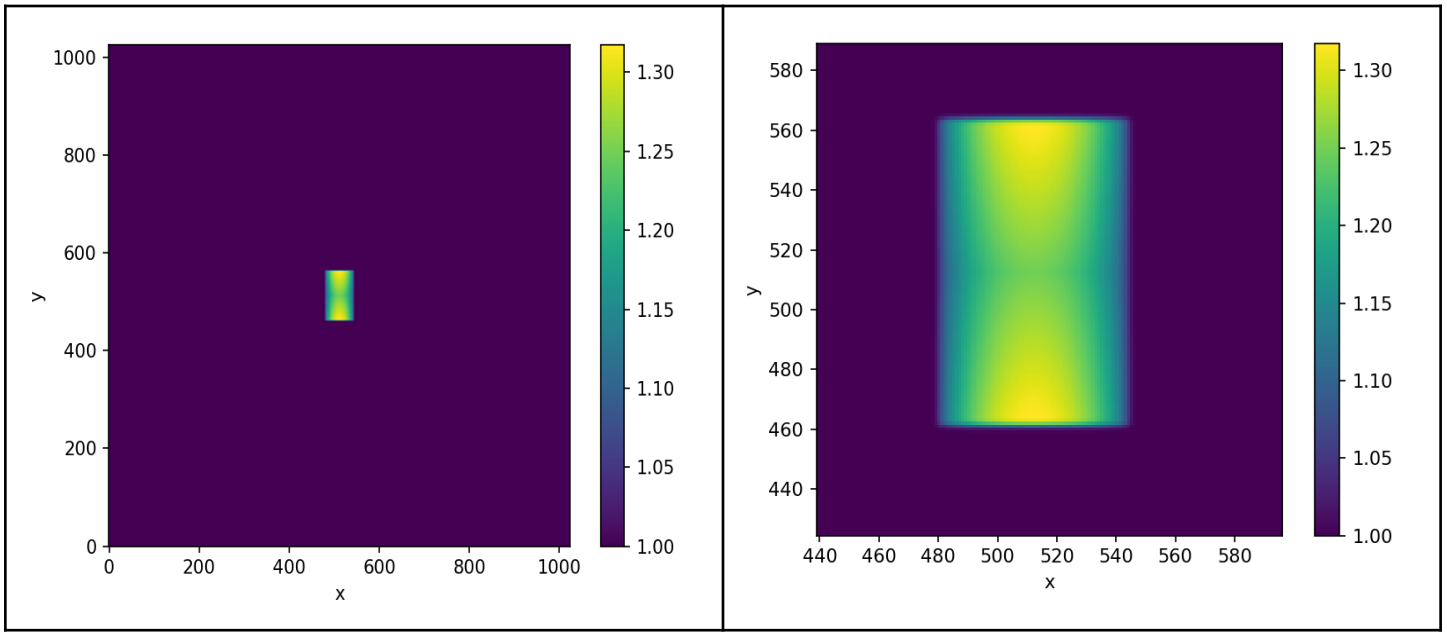


This plot looks different from what I expected… Maybe my double sigmoid function is not correct.
I will modify it.

```
14  def y_sigmoid(y):
15      σ = 0.1 * mm
16      # S = 1 / 2 * (np.tanh(-(y - h/2)/ σ) - np.tanh(-(y + h/2)/ σ))
17      S = np.abs(1 / (1 + np.exp(-(y - h/2 ) / σ)) - (1 / (1 + np.exp(-(y + h/2 ) / σ))))
18      return S # np.shape = (n_y, 1)
19
```







BAHAHA this looks the same as the previous sigmoid I made.
Ok so… there is an issue somewhere else, gotta get back to the whiteboard.

Week 6 (30/08/21 –05/09/21)

Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Meetings on **Monday 30/08** and **Wednesday 01/09**
    a. ~~Group: **2 pm**~~
    b. ~~one on one: **11 am**~~
2. **Read PyQt GUI book**
3. ~~**Broadcast TIE() into 2D**~~

30/08/21
The main issue I can see is that my TIE has not been broadcasted correctly onto 2D. I asked Chris and he has verified that I have assumed some properties of FT that are not correct. I once again need to rethink things.

$$\underbrace{\frac{\partial I}{\partial z}}_{\text{scalar}} = \underbrace{\underbrace{\nabla_T I \cdot \nabla_T \phi}_{\text{vect}\ \text{vect}} + \underbrace{I \nabla_T^2 \phi}_{\text{scalar}}}_{\text{scalar}}$$

$$\nabla_T I = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

$$\nabla_T \phi = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \phi = \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}\right)$$

$$\nabla_T I \cdot \nabla_T \phi = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \cdot \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}\right) = \frac{\partial I}{\partial x}\frac{\partial \phi}{\partial x} + \frac{\partial I}{\partial y}\frac{\partial \phi}{\partial y}$$

$$\frac{\partial I}{\partial x} = \text{ifft2}(ik_x \text{fft2}(I)) \qquad \frac{\partial I}{\partial y} = \text{ifft2}(ik_y \text{fft2}(I))$$

$$\frac{\partial \phi}{\partial x} = \text{ifft2}(ik_x \text{fft2}(\phi)) \qquad \frac{\partial \phi}{\partial y} = \text{ifft2}(ik_y \text{fft2}(\phi))$$

$$\nabla_T I \cdot \nabla_T \phi = \text{ifft2}(ik_x \text{fft2}(I))\,\text{ifft2}(ik_x \text{fft2}(\phi)) + \text{ifft2}(ik_y \text{fft2}(I))\,\text{ifft2}(ik_y \text{fft2}(\phi))$$

$$I\nabla_T^2\phi = I\nabla_T\cdot\left(\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)\phi\right) = I\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)\left(\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y}\right)$$

$$= I\left(\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}\right)$$

$$= I\ \mathrm{ifft2}\left((ik_x)^2\,\mathrm{fft2}(\phi) + (ik_y)^2\,\mathrm{fft2}(\phi)\right)$$

$$= I\ \mathrm{ifft2}\left(-k_x^2\,\mathrm{fft2}(\phi) - k_y^2\,\mathrm{fft2}(\phi)\right)$$

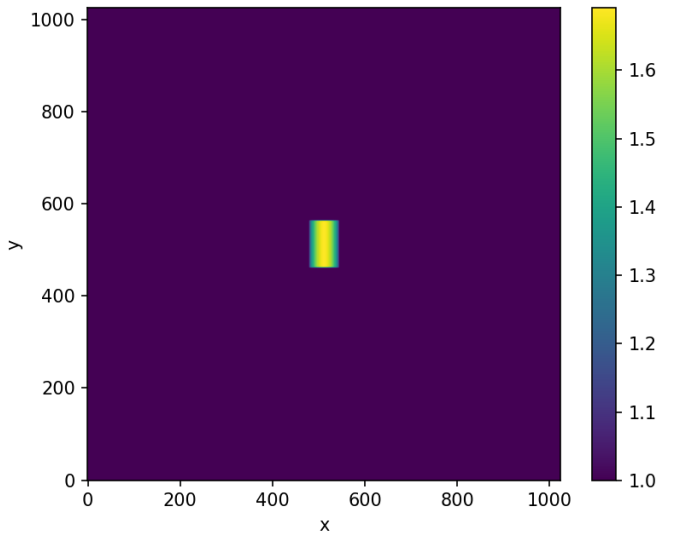
$$\frac{\partial I}{\partial z} = \mathrm{ifft2}\left(ik_x\,\mathrm{fft2}(I)\right)\mathrm{ifft2}\left(ik_x\,\mathrm{fft2}(\phi)\right) + \mathrm{ifft2}\left(ik_y\,\mathrm{fft2}(I)\right)\mathrm{ifft2}\left(ik_y\,\mathrm{fft2}(\phi)\right)$$

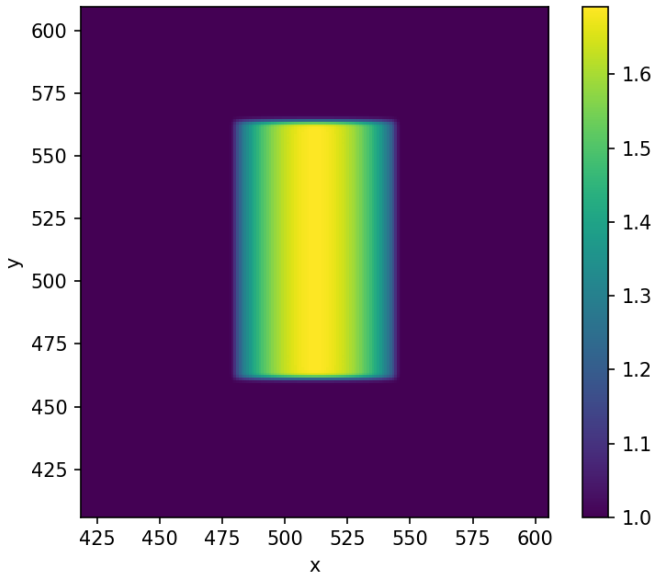
$$+ I\ \mathrm{ifft2}\left(-k_x^2\,\mathrm{fft2}(\phi) - k_y^2\,\mathrm{fft2}(\phi)\right)$$

```python
def TIE(z, I, Φ):
    '''The intensity and phase evolution of a paraxial monochromatic
    scalar electromagnetic wave on propagation (2D)'''
    FT2D_I = fft2(I)
    FT2D_Φ = fft2(Φ)
    dI_dz = (-1 / k0) * (
        np.real(
            ifft2(1j * kx * FT2D_I) * ifft2(1j * kx * FT2D_Φ)
            + ifft2(1j * ky * FT2D_I) * ifft2(1j * ky * FT2D_Φ)
            + I * ifft2(-(kx ** 2 + ky ** 2) * FT2D_Φ)
        )
    )
    return dI_dz  # np.shape(dI_dz) = (n_y, n_x)
```

Output

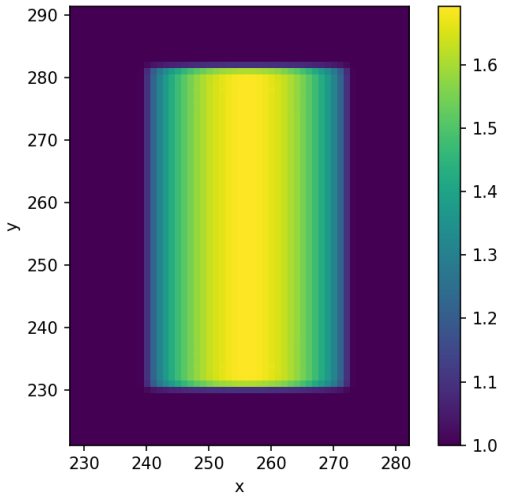
n_x = 1024
n_y = 1025





With less x and y steps...
n_x = 512
n_y = 512

```python
    # PLOTS ATTENUATION FACTOR I/I0 vs x, y (2D)
    I_list = np.load("I_list.npy") # np.shape(I_list) = (100, 512, 512)

    I = I_list[-1,:, :]
    # print(f"{np.shape(I) = }")

    plt.imshow(I / I_0, origin='lower')
    plt.colorbar()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```



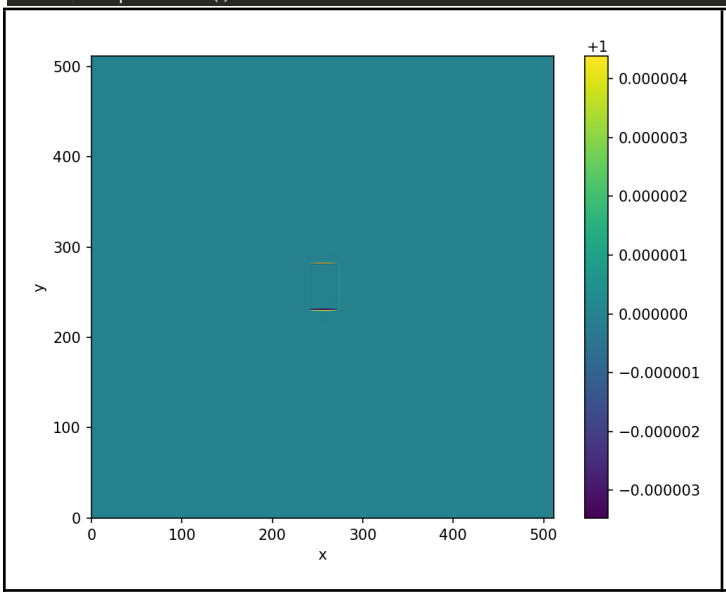
31/08/2021

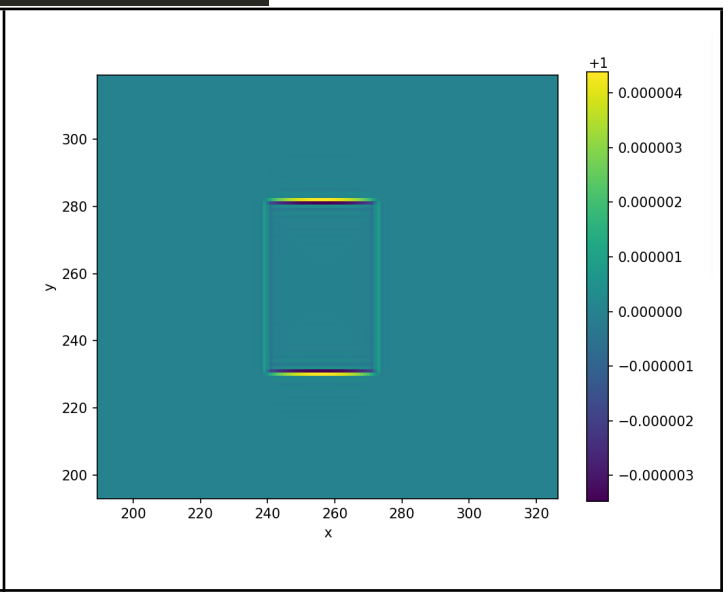
In order to visualise the phase contrast correctly I need to redefine the return call of the BLL function I as I_0 since this is in fact the IC (ie. I(z) where z = z_0) for the intensity propagated via the TIE()

```
Φ = np.load("phase_x_y.npy")
I_0 = np.load("intensity_x_y.npy")

I_list = []
while z < z_final:

    print(f"{i = }")

    # spatial evolution step
    I = Runge_Kutta(z, delta_z, I_0, Φ)
    if not i % 10:
        I_list.append(I)
    i += 1
    z += delta_z

I_list = np.array(I_list)
print(f"{np.shape(I_list) = }") #  np.shape(I_list) = (n_z / 10, n_x)

np.save(f'I_list.npy', I_list)
```

phase contrast plot

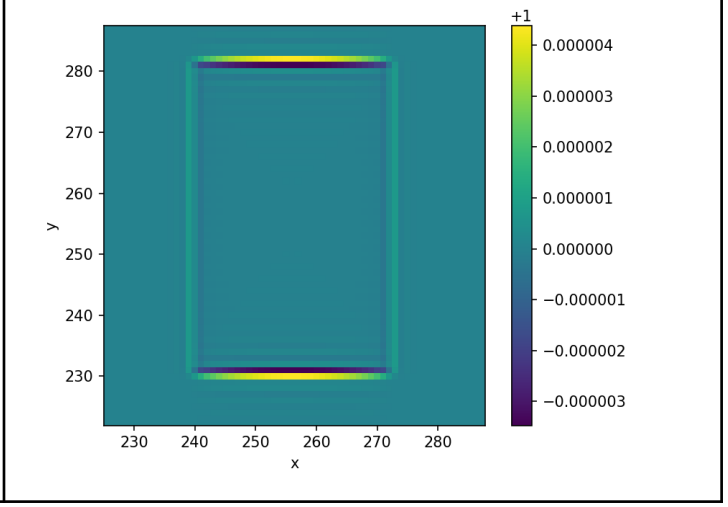
```
169     # Load file
170     I_list = np.load("I_list.npy")  # np.shape(I_list) = (n_z / 10, n_y,  n_x)
171     Φ = np.load("phase_x_y.npy")
172     I_0 = np.load("intensity_x_y.npy")
173
174     # TODO:
175     # PLOTS Phase contrast I/I0 in x, y
176     I = I_list[-1,:, :]
177     print(f"{np.shape(I) = }")
178
179     plt.imshow(I / I_0, origin='lower')
180     plt.colorbar()
181     plt.xlabel("x")
182     plt.ylabel("y")
183     plt.show()
```





What happens if I propagate the beam further after z_0? Maybe 1 m

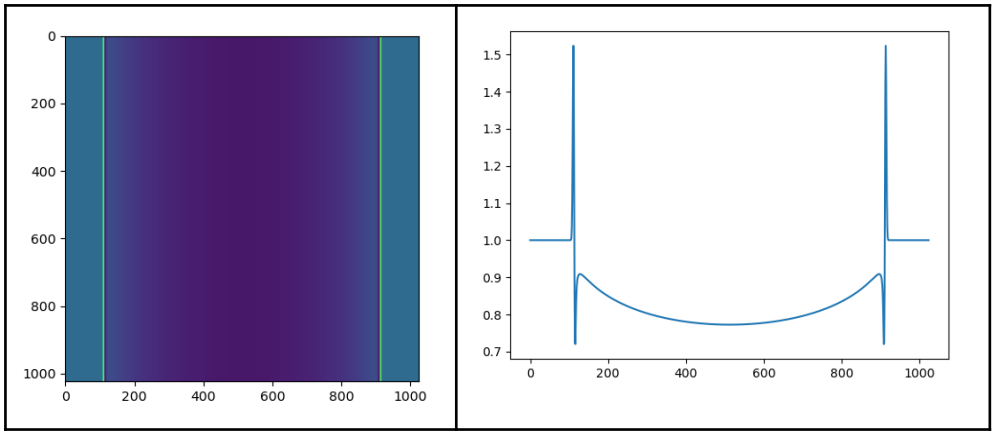
```
129     # RK Propagation loop parameters
130     i = 0
131     z = 0
132     z_final = 1000 * mm
133     delta_z = 0.1 * mm  # (n_z = 10000)
```



Not much of a change! I can see a bit of diffraction at the edges. How cool.

MK's email commentary on my latest plot

1. *the object itself looks very small compared to the number of pixels in the array so you should increase the radius*
    a. Or... decrease the x and y range (to maintain the same dimensions as the Beltran et al. 2010 paper)
2. *There seems to be some unexpected, but small, oscillations in intensity. The TIE normally only gives a single fringe pair at the boundary of such objects. These artefacts may be due to the limited object size making these effects look much larger than I'm used to.*
    a. I think I've fixed that diffraction-like artifact in the pages below
3. *Some snapshots of the phase contrast image (propagated intensity) to give you an idea of what yours should look like.*



01/09/21

MEETING DAY!

Questions for MK:

1. Once you've been added you should be able to navigate from your personal profile to **Groups -> xrayimaging -> Shared Projects** and see the project.
   Access is now granted
2. Why are my peaks so small?
   No idea
3. Clarify with MK the axis labels in his examples
   Horizontal axis is x, vertical axis is I

Bug fix

My Phase contrast looks opposite to MK's… mine is bright and his is dark?
Looking into it I noticed that I cancelled out a negative sign in the BLL() function.
Specifically I had line 61 set with a negative sign after the += and also line 62 as is. **Therefore my intensity was not being attenuated!**

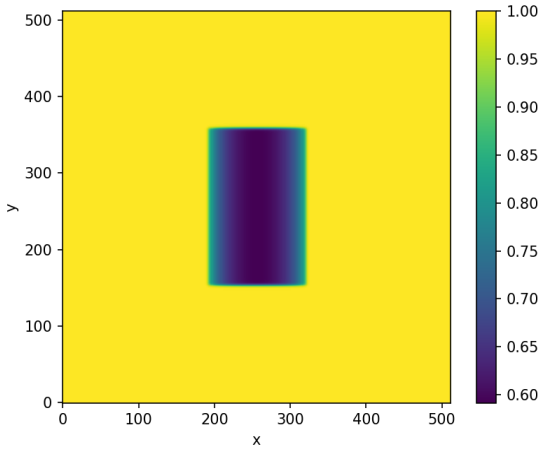
```
53  def BLL(x, y):
54      # TIE IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
55      z = np.linspace(-2 * R, 2 * R, 512, endpoint=False)
56      dz = z[1] - z[0]
57      # Euler's method
58      F = np.zeros_like(x * y)
59      for z_value in z:
60          print(z_value)
61          F += μ(x, y, z_value) * dz
62      I = np.exp(- F) * I_0
63      return I # np.shape(I_0) = (n_y, n_x)
```

I also made some other changes, first I decreased the x and y ranges

```
87  def globals():
88      # x-array parameters
89      n = 512
90
91      n_x = n
92      x_max = 25 * mm
93      x = np.linspace(-x_max, x_max, n_x, endpoint=False)
94      delta_x = x[1] - x[0]
95      size_x = x.size
96
97      # y-array parameters
98      n_y = n
99      y_max = 25 * mm
00      y = np.linspace(-y_max, y_max, n_y, endpoint=False).reshape(n_y, 1)
01      delta_y = y[1] - y[0]
02      size_y = y.size
```

Then I increased z_final but decreased the number of steps

```
129     # RK Propagation loop parameters
130     i = 0
131     z = 0
132     z_final = 1 * m
133     delta_z = 1 * mm  # (n_z = 1000)
```

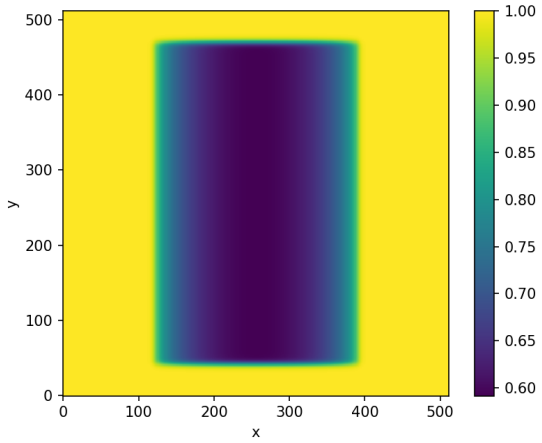


I can still make the x and y ranges smaller…

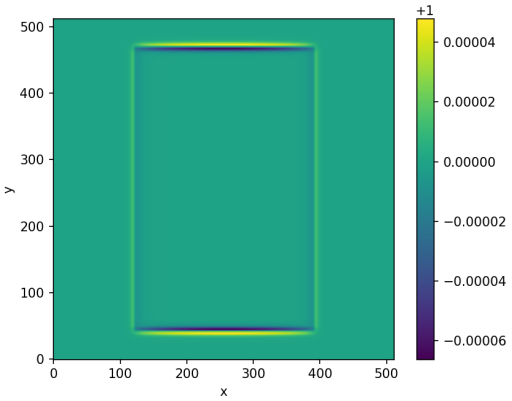
```
179     plt.imshow(I, origin='lower')
180     plt.colorbar()
181     plt.xlabel("x")
182     plt.ylabel("y")
183     plt.show()
```

I'll make the y_max and x_max values 12 mm

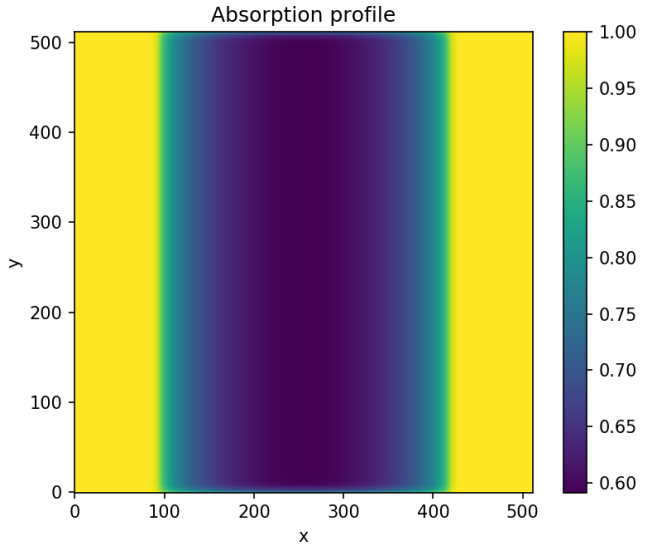
For x_max = y_max = 12 mm

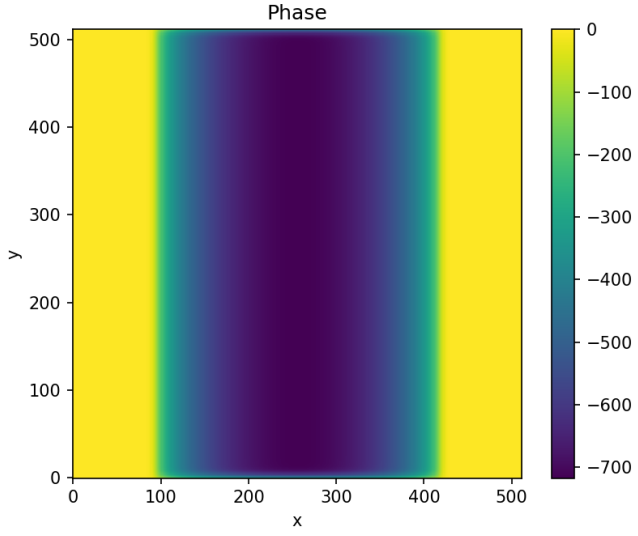


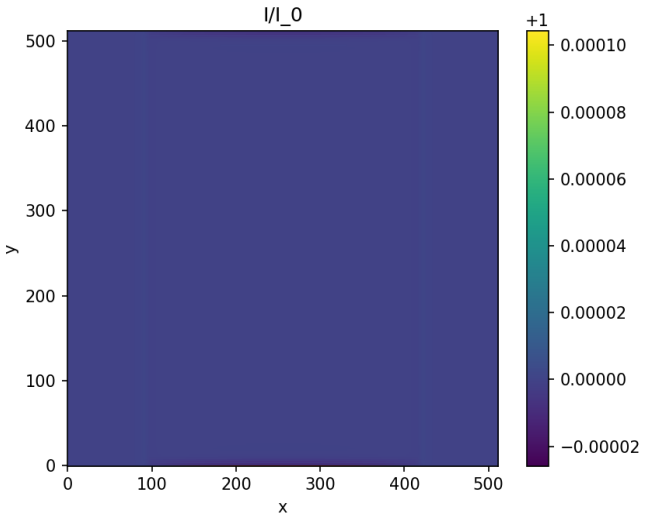
I/I_0



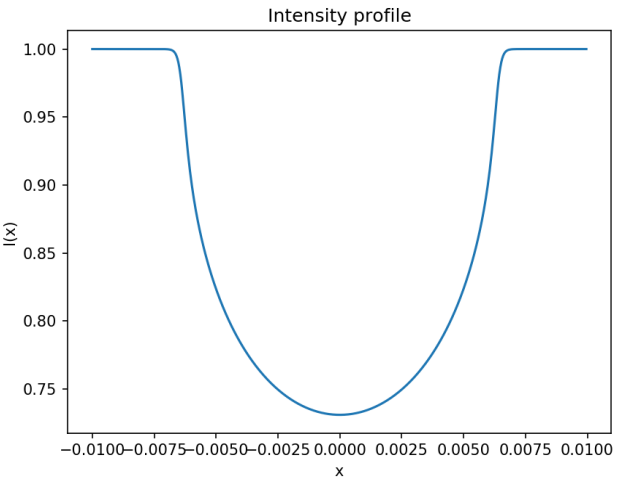
For x_max = y_max = 10 mm



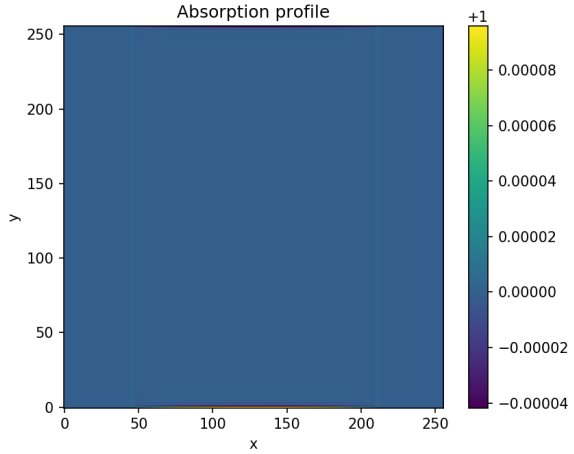




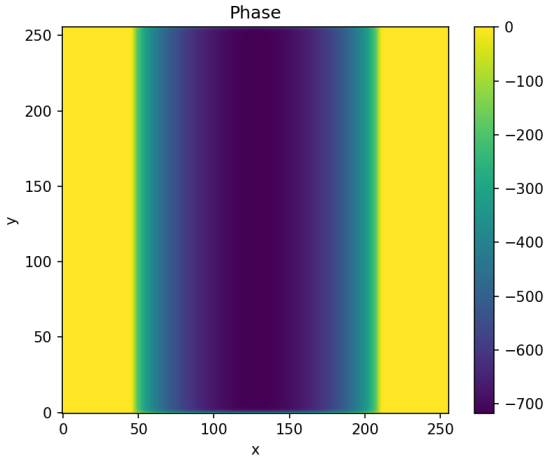
huh?

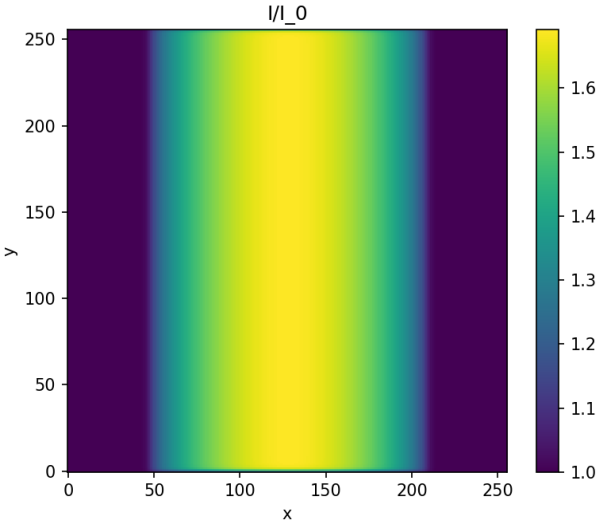


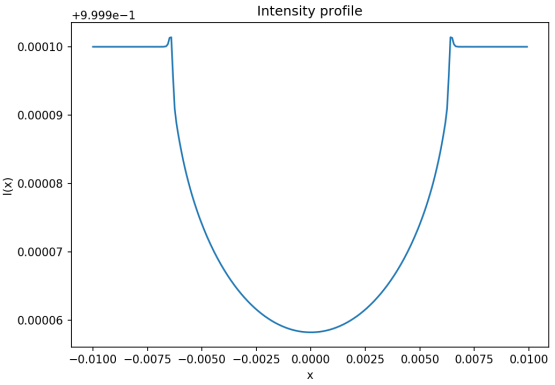
Maybe my blurring of the edges is too strong!
$\sigma = 0.05$mm for both μ(x, y, z) and δ(x, y, z)
Here I decreased n_all



HUH? This looks very wrong







This profile makes me hopeful...
I'll decrease the size of σ even further!

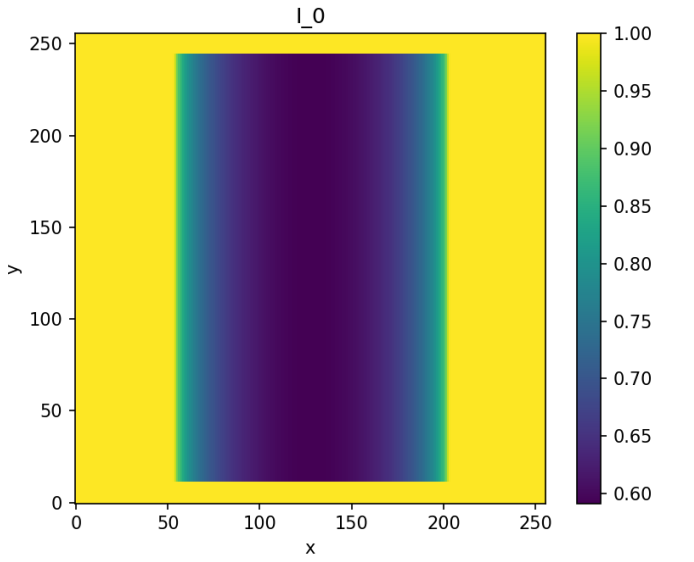
OMG This is the best The problem was that I made a silly bug!
I had mistakenly called my RK algorithm with the initial condition I_0 over and over… instead of redefining and evolving an advanced version of I

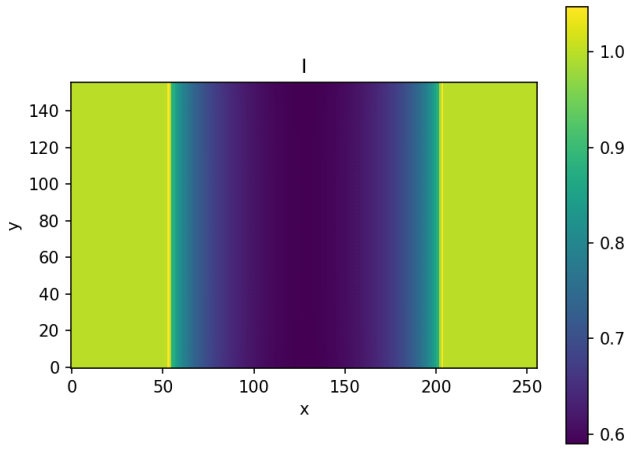
Current code status

```python
def y_sigmoid(y):
    σ = 0.004 * mm
    S = np.abs(1 / (1 + np.exp(-(y - h/2) / σ)) - (1 / (1 + np.exp(-(y + h/2) / σ))))
    return S # np.shape = (n_y, 1)


def δ(x, y, z):
    '''Refractive index: δ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    δ0 = 462.8 * nm
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.03 * mm
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
    return δ_array # np.shape(δ_array) = (n_y, n_x)


def μ(x, y, z):
    '''attenuation coefficient: μ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    μ0 = 41.2 # per meter
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    σ = 0.03 * mm
    μ_array = μ0 * (1 / (1 + np.exp((r - R) / σ))) * y_sigmoid(y)
    return μ_array # np.shape(μ_array) = (n_y, n_x)


def phase(x, y):
    # phase gain as a function of the cylinder's refractive index
    z = np.linspace(-2 * R, 2 * R, 1024, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    ϕ = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        ϕ += -k0 * δ(x, y, z_value) * dz
    return ϕ # np.shape(ϕ) = (n_y, n_x)


def BLL(x, y):
    # TIE IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
    z = np.linspace(-2 * R, 2 * R, 1024, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    F = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        F += μ(x, y, z_value) * dz
    I = np.exp(- F) * I_initial
    return I # np.shape(I) = (n_y, n_x)


def TIE(z, I, ϕ):
    '''The intensity and phase evolution of a paraxial monochromatic
    scalar electromagnetic wave on propagation (2D)'''
    FT2D_I = fft2(I)
    FT2D_ϕ = fft2(ϕ)
    dI_dz = (-1 / k0) * (
        np.real(
            ifft2(1j * kx * FT2D_I) * ifft2(1j * kx * FT2D_ϕ)
            + ifft2(1j * ky * FT2D_I) * ifft2(1j * ky * FT2D_ϕ)
            + I * ifft2(-(kx ** 2 + ky ** 2) * FT2D_ϕ)
        )
    )
    return dI_dz  # np.shape(dI_dz) = (n_y, n_x)


def Runge_Kutta(z, delta_z, I, ϕ):
    # spatial evolution 4th order RK
    # z is single value, delta_z is step
    k1 = TIE(z, I, ϕ)
    k2 = TIE(z + delta_z / 2, I + k1 * delta_z / 2, ϕ)
    k3 = TIE(z + delta_z / 2, I + k2 * delta_z / 2, ϕ)
    k4 = TIE(z + delta_z, I + k3 * delta_z, ϕ)
    return I + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)  # shape = (n_y, n_x)
```

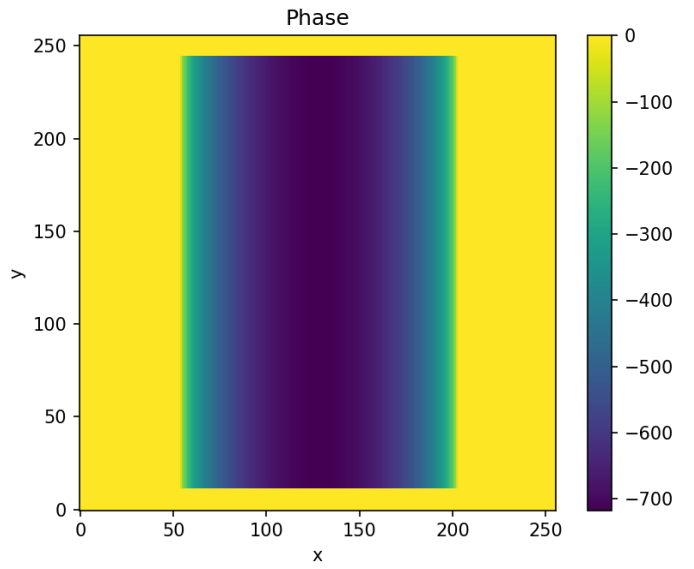
```python
    return I_z


def globals():
    # x-array parameters
    n_all = 256

    n_x = n_all
    x_max = 11 * mm
    x = np.linspace(-x_max, x_max, n_x, endpoint=False)
    delta_x = x[1] - x[0]
    size_x = x.size

    # y-array parameters
    n_y = n_all
    y_max = 11 * mm
    y = np.linspace(-y_max, y_max, n_y, endpoint=False).reshape(n_y, 1)
    delta_y = y[1] - y[0]
    size_y = y.size

    # X-ray beam parameters
    λ = 0.05166 * nm  # x-rays wavelength
    k0 = 2 * np.pi / λ  # x-rays wavenumber

    # Cylinder parameters
    D = 12.75 * mm
    R = D / 2
    z_c = 0 * mm
    x_c = 0 * mm
    h = 20 * mm

    # For Fourier space
    kx = 2 * np.pi * np.fft.fftfreq(size_x, delta_x)
    ky = 2 * np.pi * np.fft.fftfreq(size_y, delta_y).reshape(size_y, 1)

    return n_x, n_y, x, y, k0, R, z_c, x_c, h, kx, ky
```

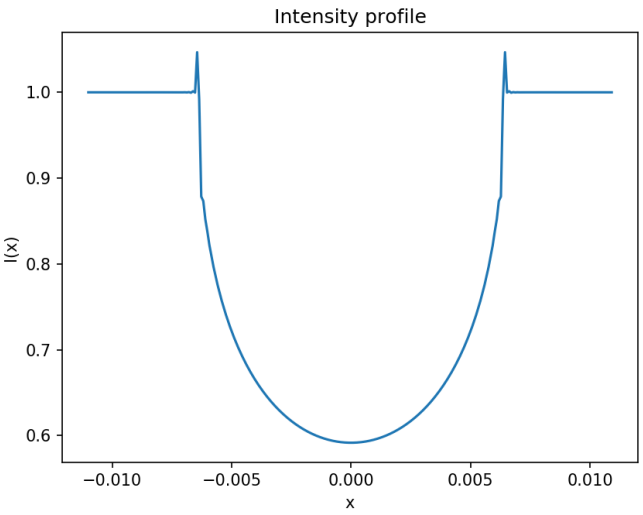
$\sigma = 0.03$mm for both μ(x, y, z) and δ(x, y, z) x_max = y_max = 11 * mm	



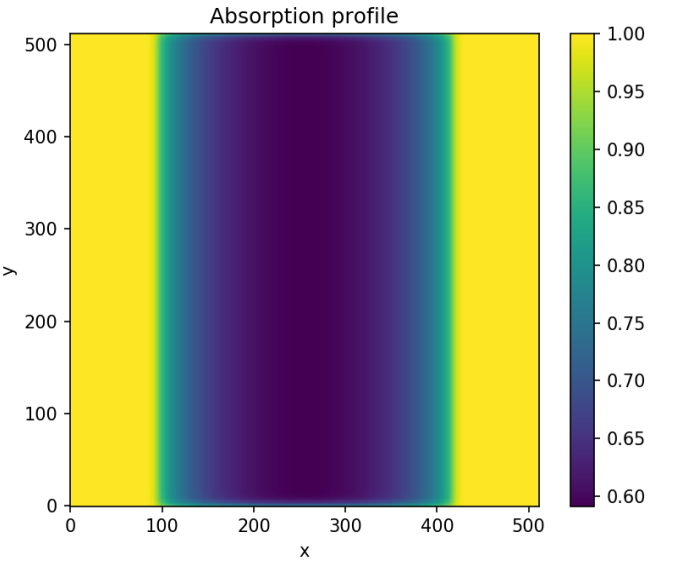


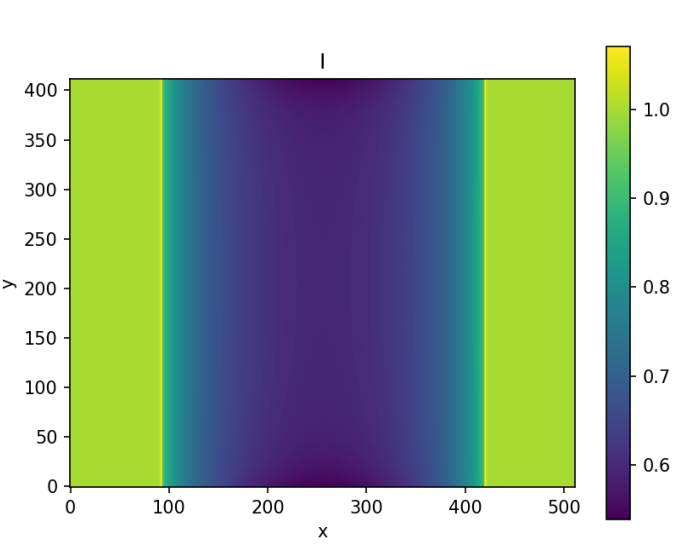
I decreased the number of displayed pixels in the y direction

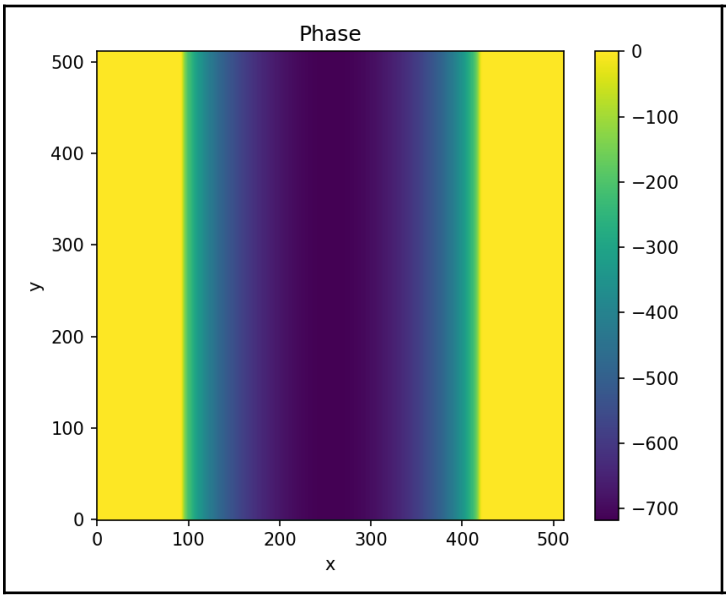


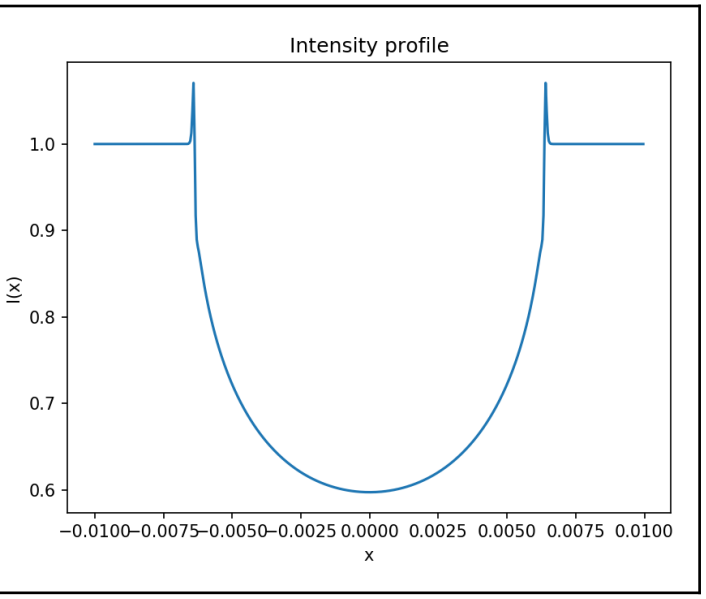


$\sigma = 0.03$mm for both μ(x, y, z) and δ(x, y, z)
x_max = y_max = 10 * mm
N_x = n_y = 512









02/09/21

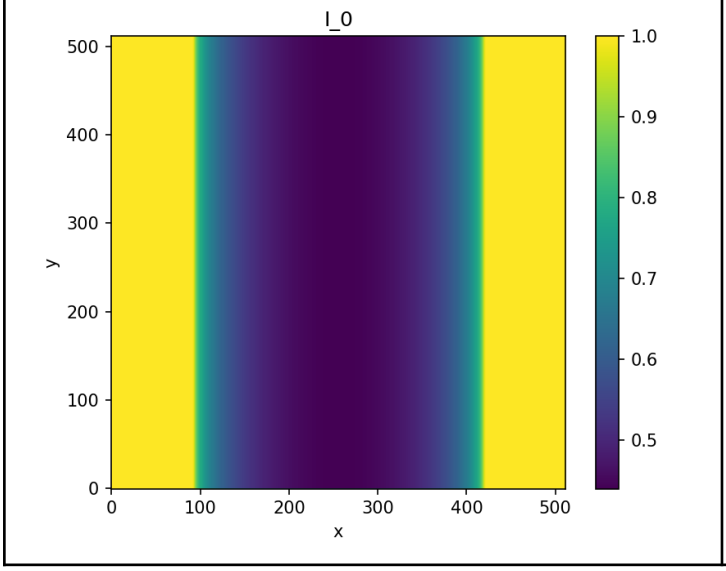
Using MK's simulation parameters

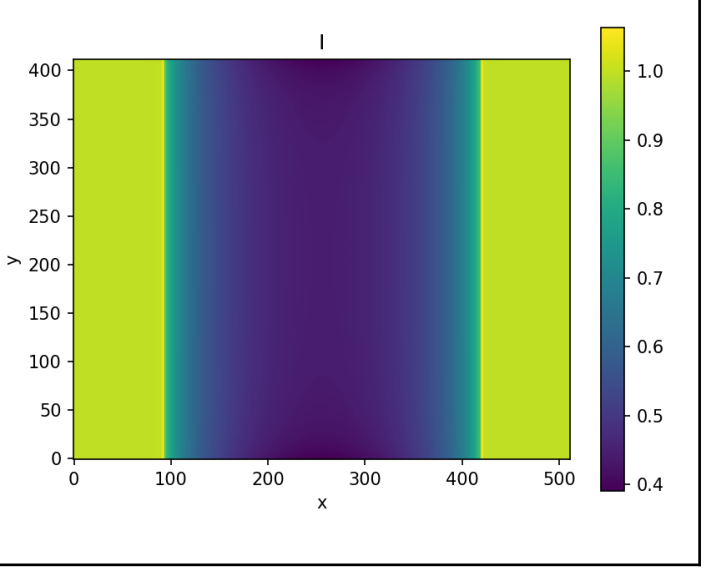
I originally chose my X-ray wavelength based on the parameters provided in (Beltran et al. 6430) . But in order to repeat the results in MK's simulation I should probably use his parameters.
These values are from his script **"Energy_Dispersion_Sim-1.py"**

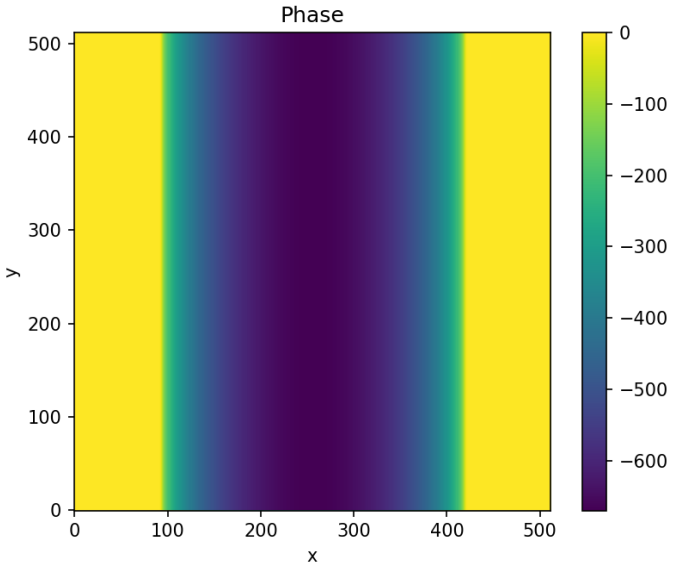
$\sigma = 0.03$mm for both μ(x, y, z) and δ(x, y, z)
x_max = y_max = 10 * mm

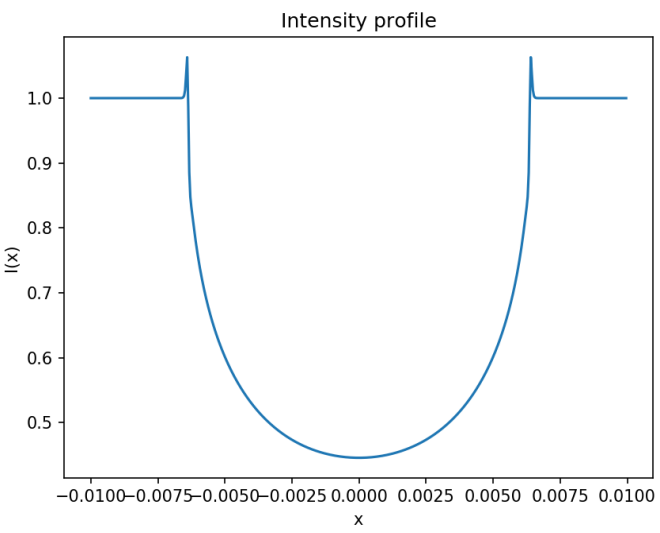
Old parameters
δ0 = 462.8 * nm
μ0 = 41.2 # per meter
energy = 24 * keV
λ = 0.05166 * nm

New parameters
energy1 = 3.5509e-15 * J # =22.1629 #keV - Ag k-alpha1
λ = 0.055942 * nm
delta1=4.68141e-07
mu1=64.38436
My old parameters and MK's parameters are very similar though



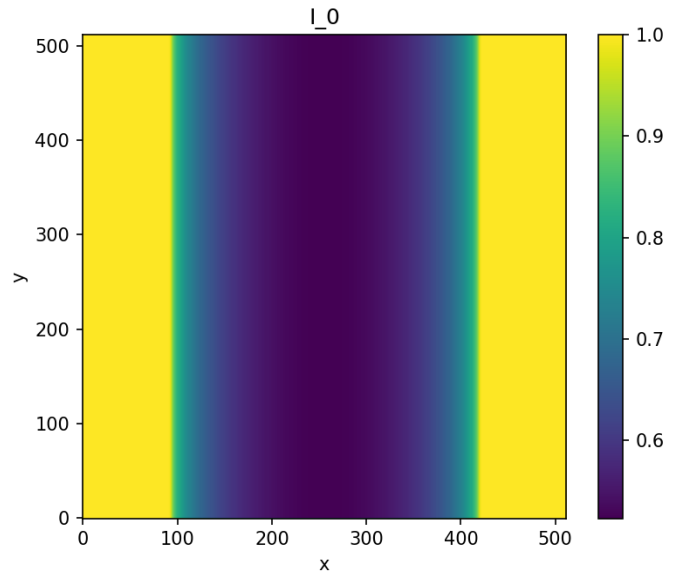


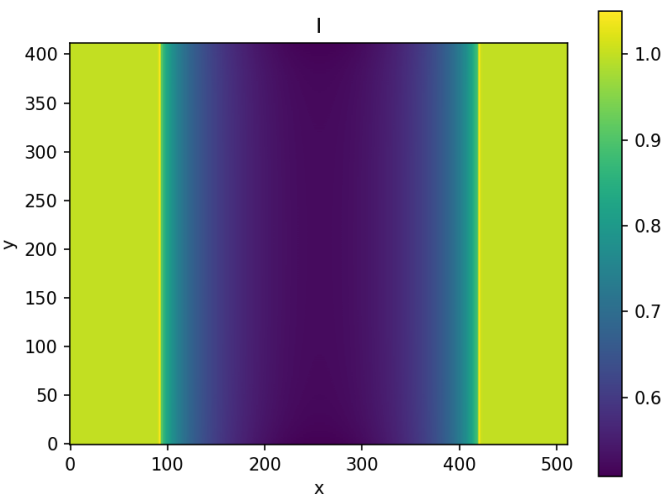


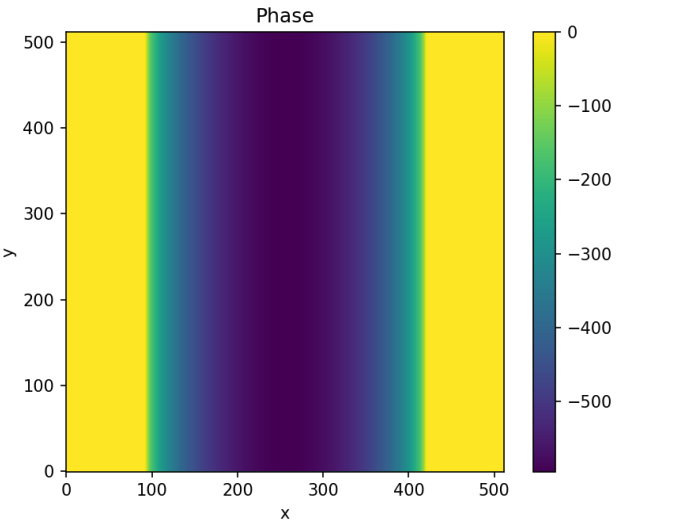


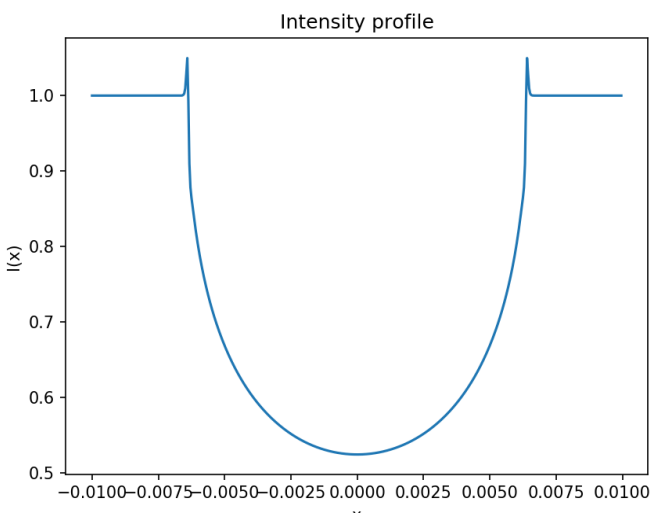
OTHER MK parameters

energy2 = 24.942 #keV - Ag k-beta1
λ = 0.04971 * nm
delta2=3.69763e-07
mu2=50.9387









From these results I can see that the potential issues my code faces are due to

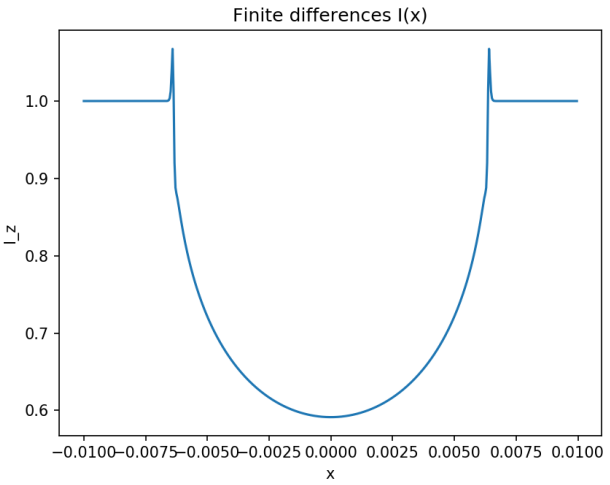
1. **TIE()**
   With MK's help I made this first order FD solver:

```python
def finite_diff(z, I):
    # first order finite differences
    I_z = I + z * TIE(z, I, Φ)
    return I_z
```

The function returns an array that can be plotted having y as a constant. The Intensity profile looks very very similar to the propagated intensity profile I've obtained with my RK algorithm… The main thing in common between these functions is the TIE.

```python
I_0 = BLL(x, y)
# np.save(f'intensity_x_y.npy', I)
```

```python
# Finite differences 1st order
I_z = finite_diff(1 * m, I_0)
# PLOT I_z vs x (a single slice)
plt.plot(x, I_z[np.int(n_y / 2),:])
plt.xlabel("x")
plt.ylabel("I_z")
plt.title("Finite differences I(x)")
plt.show()
```

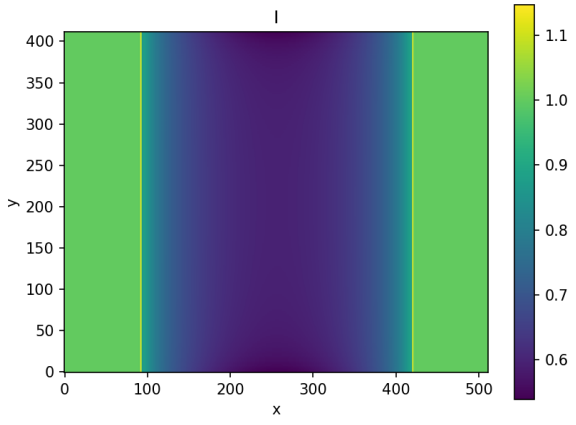


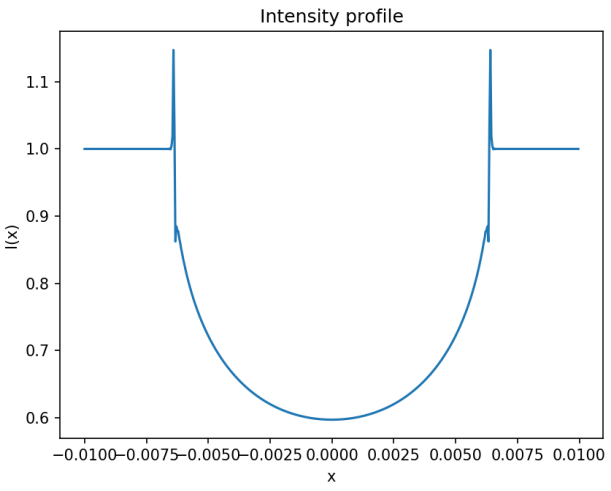
What does the TIE() use?

δ(x, y, z) ⟵Can Increase resolution in x and y to test if σ **increases** the contrast further!

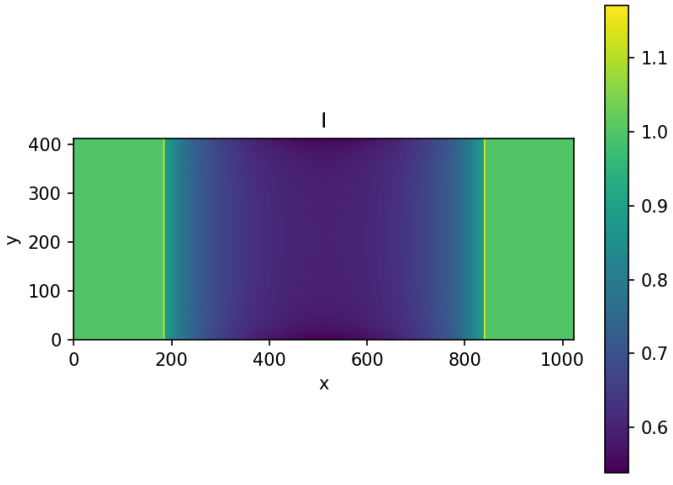
μ(x, y, z) ⟵

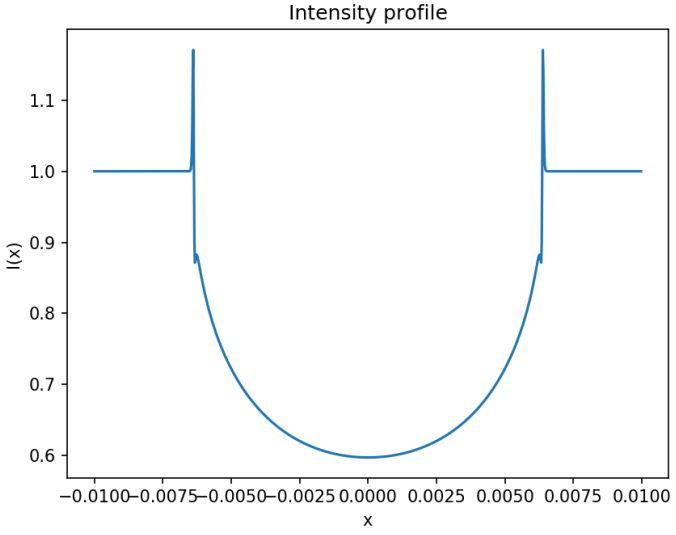
I decreased σ **to 0.018 * mm** in both δ(x, y, z) and μ(x, y, z)



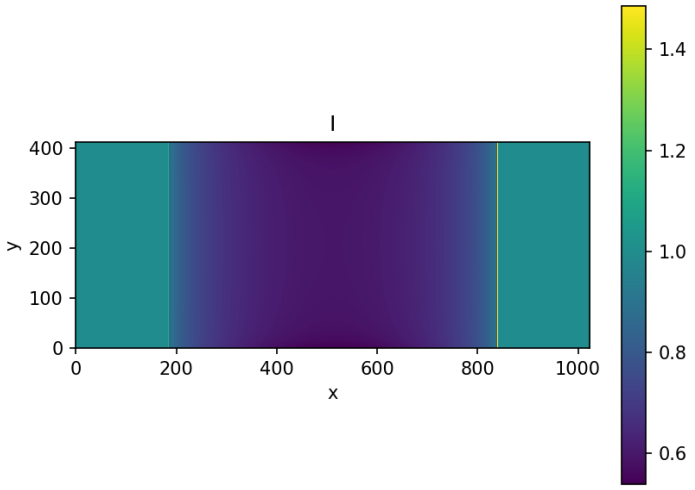


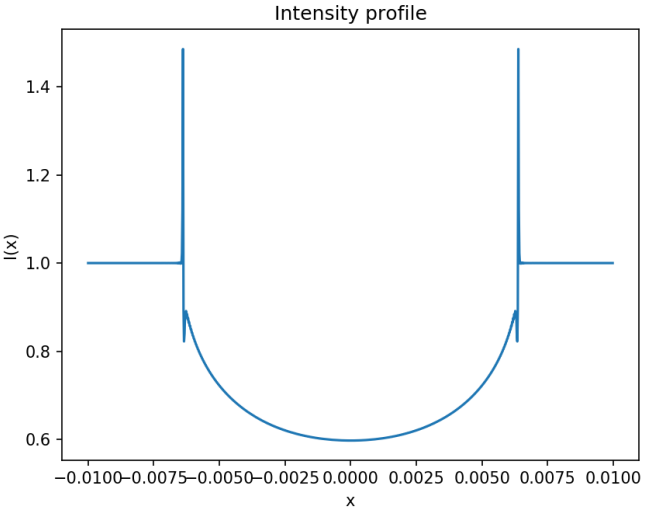
I will now increase the number of points in x to **n_x = 1024** while keeping everything else equal





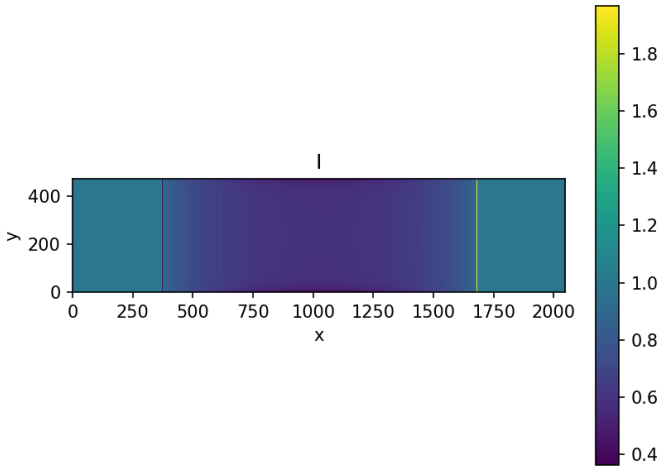
I will now decrease σ **to 0.01 * mm** in both δ(x, y, z) and μ(x, y, z)
And keep all else equal

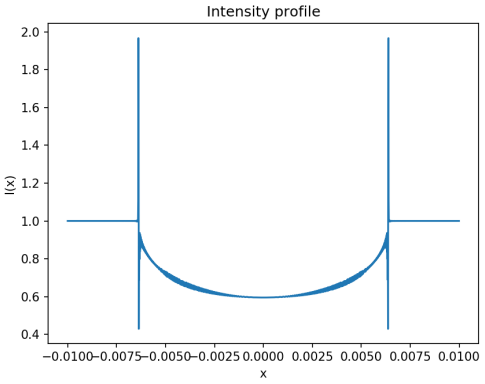




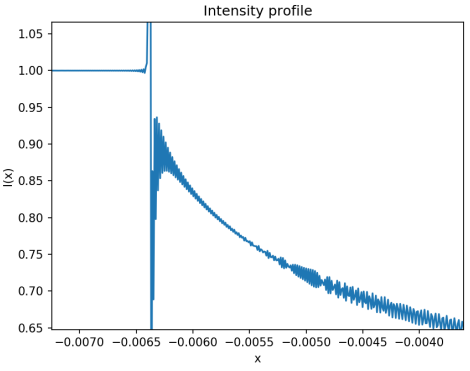
It looks like the asymmetry is due to my computer not displaying the image with enough pixels… like the pixels are getting averaged out by their surroundings and therefore undergoing **aliasing.**
.https://en.wikipedia.org/wiki/Aliasing

I will now decrease σ **to 0.005 * mm** in both δ(x, y, z) and μ(x, y, z), double **n_x to 2048** and increase the size of my **delta_z to 10 * mm.**





WOW! Yay this worked.. But things are exploding a bit



My code is very slow at the moment so that makes it a pain to check. I know it's better to make sure code works before optimisation can happen but I think I can do better than the time I am doing at the moment.

```
63  def gradΦ_laplacianΦ(Φ):
64      FT2D_Φ = fft2(Φ)
65      dΦ_dx = ifft2(1j * kx * FT2D_Φ)
66      dΦ_dy = ifft2(1j * ky * FT2D_Φ)
67      lap_Φ = ifft2(-(kx ** 2 + ky ** 2) * FT2D_Φ)
68      return dΦ_dx, dΦ_dy, lap_Φ
```

With this function I save roughly half the time spent in each computation of the TIE… I pre-calculate all derivatives of Φ!

The phase velocity

If I find out how fast my waves evolve in the TIE then I can adjust the z-evolution step to match the nyquist mode!

$$\frac{\partial I}{\partial z} = \frac{1}{k_0}\left(\frac{d}{dx}I \cdot \frac{d}{dx}\phi + I\frac{d^2}{dx^2}\phi\right)$$

Since the TIE is isotropic, i'll focus on x.

$$\frac{\partial I}{\partial z} = \frac{1}{k_0}\left(\frac{d}{dx}I \cdot \frac{d}{dx}\phi + I\frac{d^2}{dx^2}\phi\right)$$

evolution occurs along z (ie. z is t essentially)

$$\therefore \frac{\partial I}{\partial t} = \frac{1}{k_0}\left(\frac{d}{dx}I \cdot \frac{d}{dx}\phi + I\frac{d^2}{dx^2}\phi\right)$$

let $I = e^{i(kx-\omega t)}$

$$\frac{\partial I}{\partial t} = -i\omega I \quad \& \quad \frac{d}{dx}I = ikI$$

$$\therefore -i\omega I = \frac{1}{k_0}\left(ikI \cdot \frac{d}{dx}\phi + I\frac{d^2}{dx^2}\phi\right)$$

factor out I & let $f(x) = \frac{d}{dx}\phi$ & $g(x) = \frac{d^2}{dx^2}\phi$

$$\therefore -i\omega = \frac{1}{k_0}(ikf(x) + g(x))$$

$$\therefore \omega = \frac{-1}{k_0}\left(kf(x) + \frac{1}{i}g(x)\right)$$

divide by k

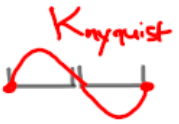
$$\therefore \frac{\omega}{k} = \frac{-1}{k_0}\left(f(x) + \frac{1}{ik}g(x)\right)$$

since $\frac{\omega}{k} = v_p$

$$v_p = \frac{-1}{k_0}\left(f(x) + \frac{1}{ik}g(x)\right)$$

Phase Velocity

nyquist mode



smallest mode that can be represented in my grid

I need:

$$\frac{\Delta x}{\Delta z} << v_p(k_{nyquist})$$

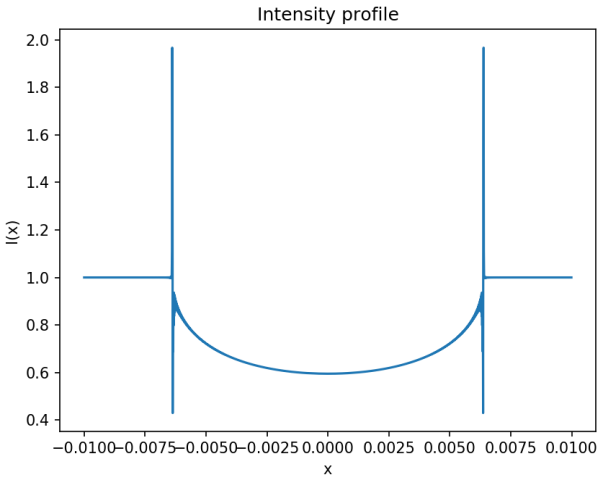
Whoops I got a negative sign wrong there! Whatever, since it doesn't affect the argument in theory.

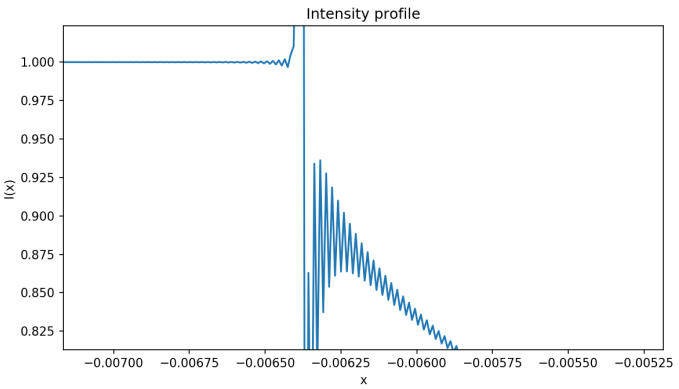
Testing

I'm gonna decrease dz by doubling the number of points in the z_arrays in δ and μ z = np.linspace(-2 * R, 2 * R, **1024**, endpoint=False) to

```
x = np.linspace(-x_max, x_max, 2048, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 4096, endpoint=False)
```

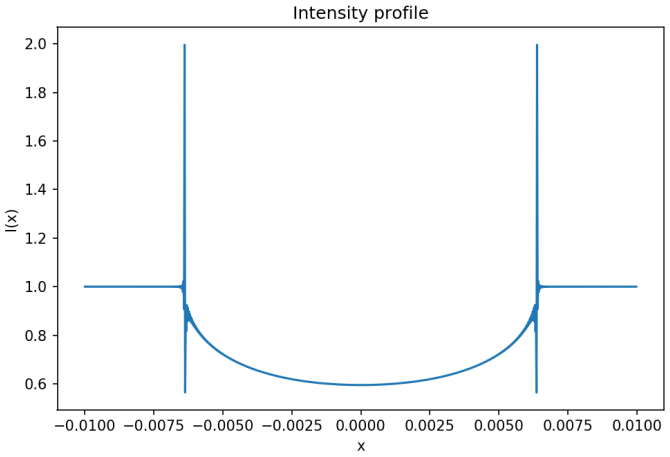
dx/dz = 1/2

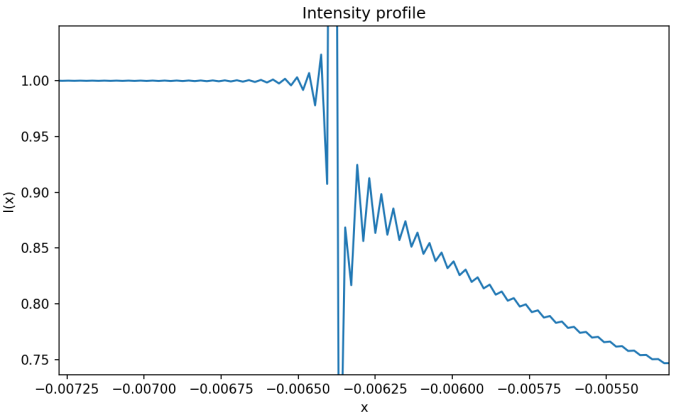




x = np.linspace(-x_max, x_max, **1024**, endpoint=False)
z = np.linspace(-2 * R, 2 * R, **8192**, endpoint=False)

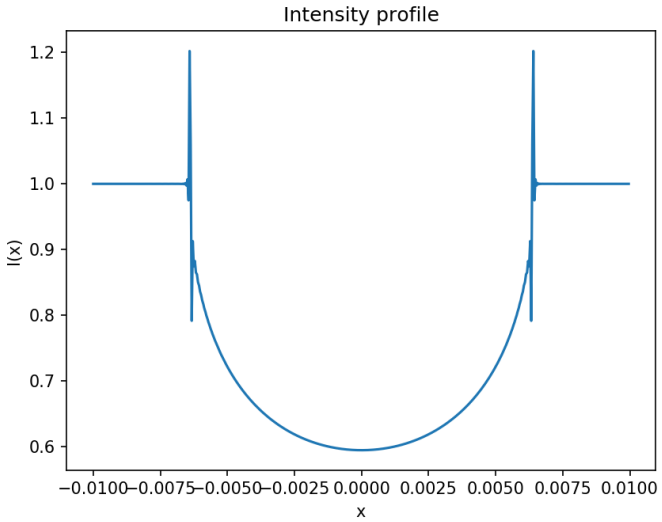
dx/dz = 1/8

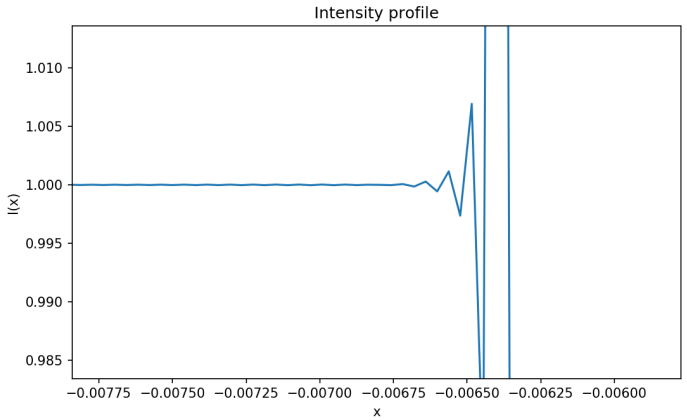




x = np.linspace(-x_max, x_max, **512**, endpoint=False)
z = np.linspace(-2 * R, 2 * R, **8192**, endpoint=False)

dx/dz = 1/16

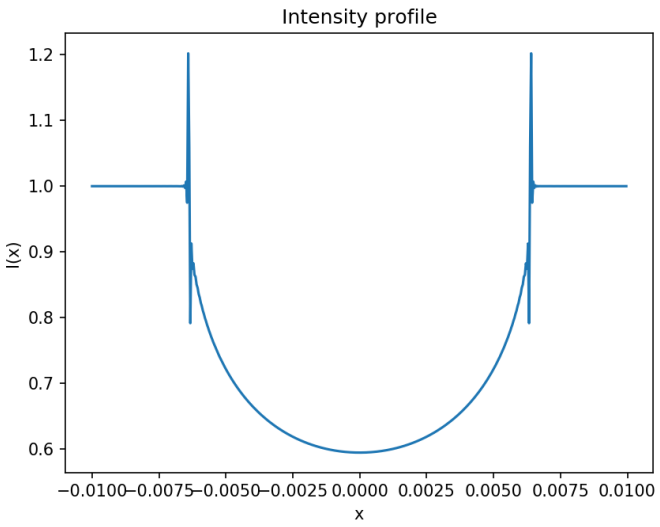


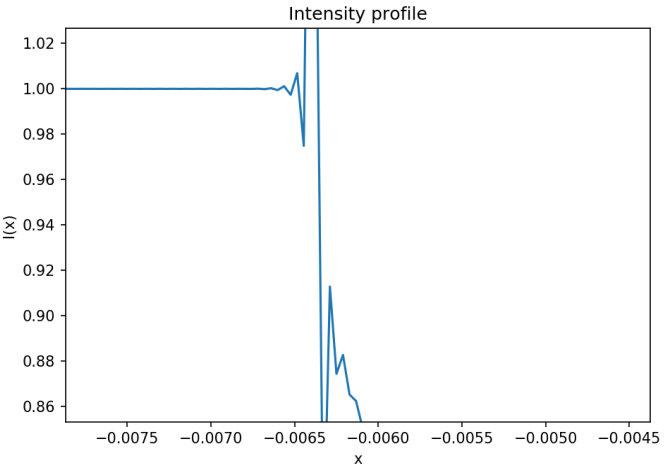


Note that the contrast fringes are much lower!

x = np.linspace(-x_max, x_max, **512**, endpoint=False)
z = np.linspace(-2 * R, 2 * R, **32768**, endpoint=False)

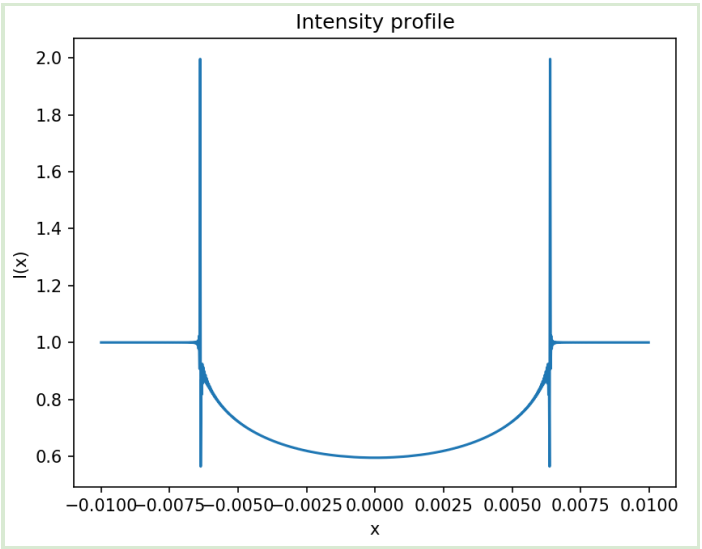
dx/dz = 1/64

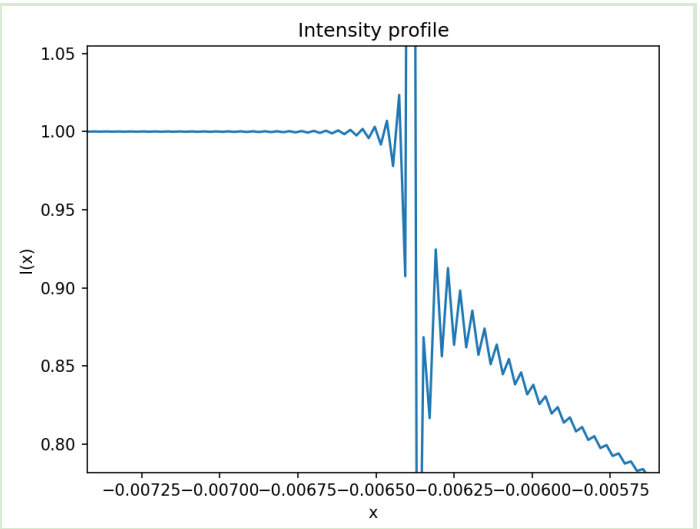




x = np.linspace(-x_max, x_max, 1024, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 65536, endpoint=False)
σ = **0.005 * mm**

dx/dz = 1/64

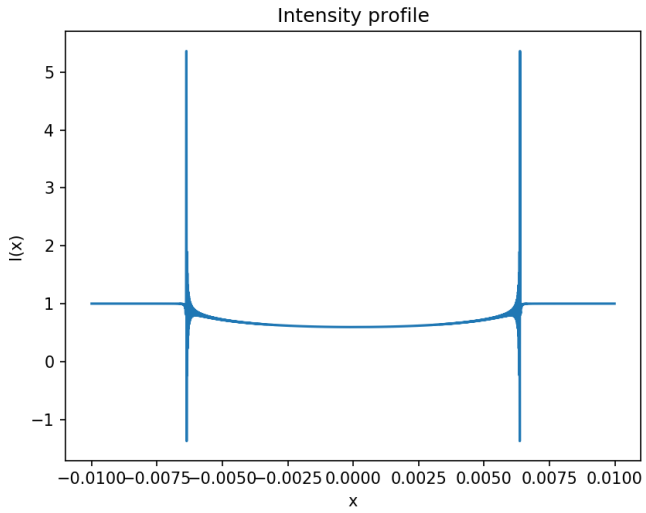


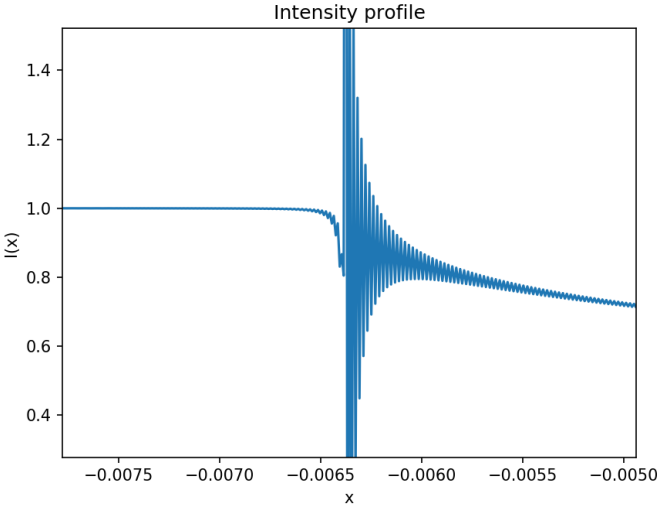


Note that the contrast fringes are much higher again!
I think that this is probably the minimum ideal size for n_x

x = np.linspace(-x_max, x_max, 2048, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 65536, endpoint=False)
σ = 0.0025 * mm

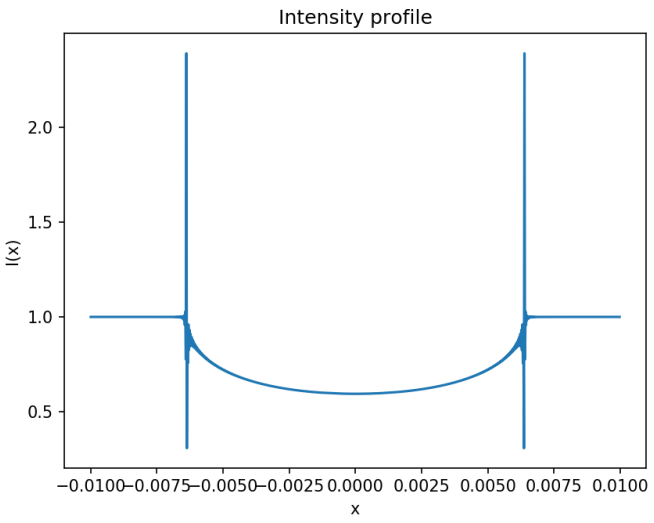
dx/dz = 1/32

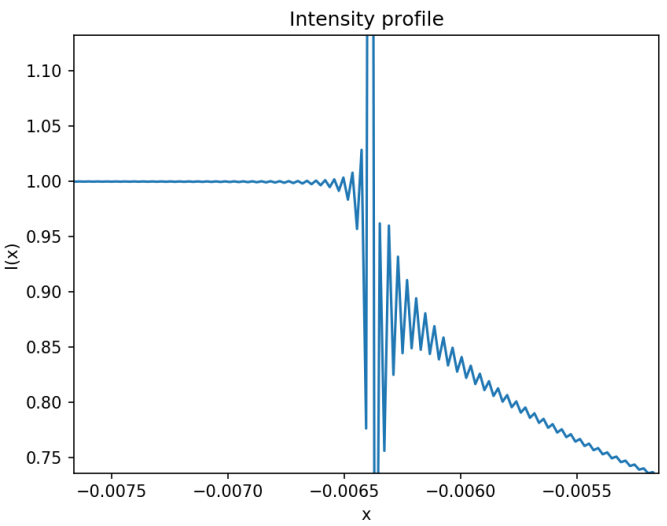




x = np.linspace(-x_max, x_max, 1024, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 65536, endpoint=False)
σ = 0.0025 * mm

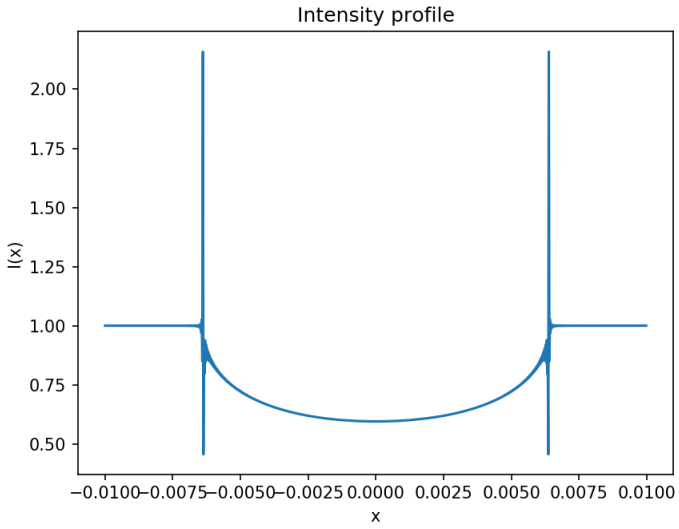
dx/dz = 1/64

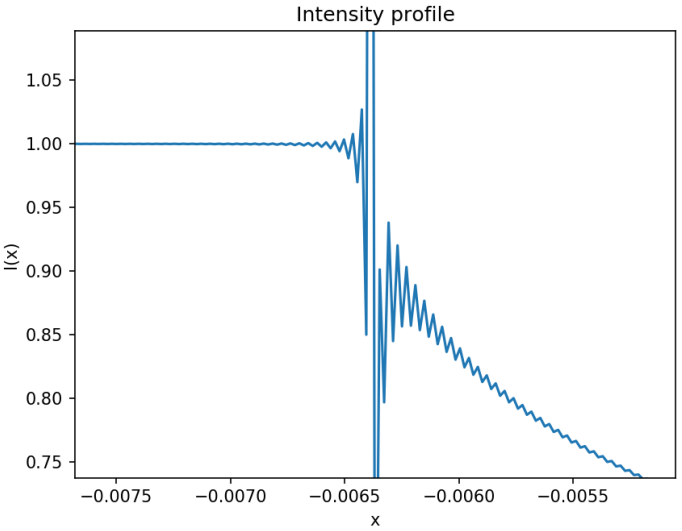




x = np.linspace(-x_max, x_max, **1024**, endpoint=False)
z = np.linspace(-2 * R, 2 * R, **65536**, endpoint=False)
From σ = **0.004 * mm**

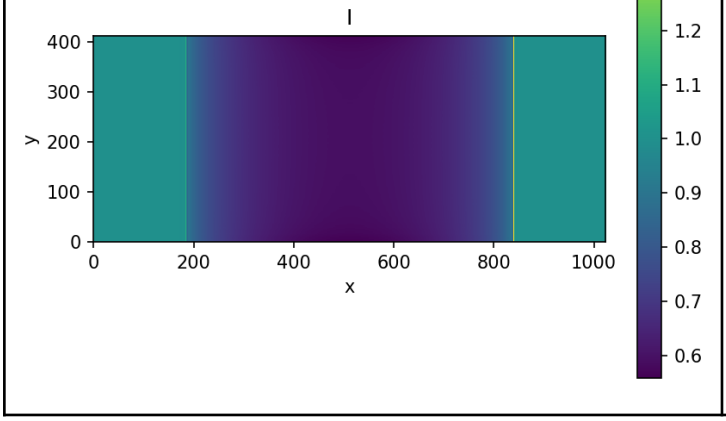
dx/dz = 1/64

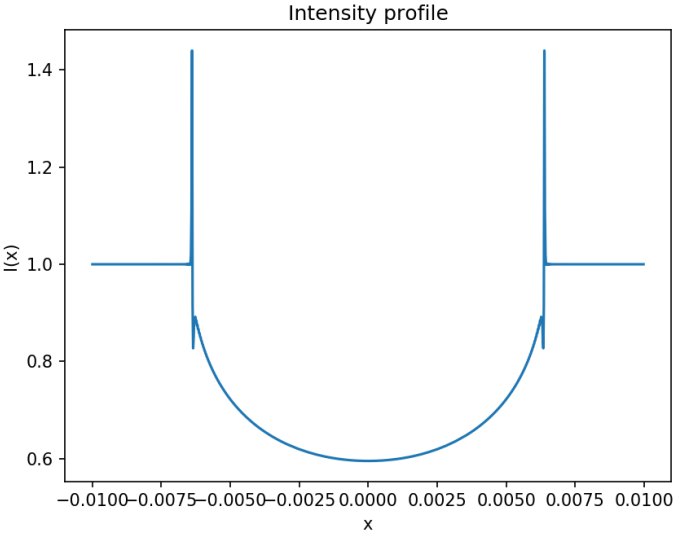




Current output

x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
y= np.linspace(-10*mm, 10*mm, **512,** endpoint=False)
z = np.linspace(-2 * R, 2 * R, **65536**, endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min})) = 0.5$





So it looks like I cannot have my cake and eat it too. Increasing σ so the blurring covers more pixels does not make my Intensity profile match MK's. I am not sure why the equation can't be resolved in a stable and accurate way even though I have a higher order propagation algorithm and a pretty good way of calculating the derivatives.
05/09/21

Context switching

Since I haven't been able to make my method work to obtain a fully stable solution, I decided to change my approach and use a more similar methodology to the xri project used by the X-ray group.

```python
def gradΦ_laplacianΦ(Φ):
    dΦ_dx = np.gradient(Φ, delta_x, axis=1)
    dΦ_dy = np.gradient(Φ, delta_y, axis=0)
    lap_Φ = laplace(Φ / delta_x**2)
    return dΦ_dx, dΦ_dy, lap_Φ


def TIE(z, I):
    '''The intensity and phase evolution of a paraxial monochromatic
    scalar electromagnetic wave on propagation (2D)'''
    dI_dx = np.gradient(I, delta_x, axis=1)
    dI_dy = np.gradient(I, delta_y, axis=0)
    dI_dz = (-1 / k0) * (
        dI_dx * dΦ_dx +
        dI_dy * dΦ_dy +
        I * lap_Φ
        )
    return dI_dz  # np.shape(dI_dz) = (n_y, n_x)


def finite_diff(z, I):
    # first order finite differences
    I_z = I + z * TIE(z, I)
    return I_z


def Runge_Kutta(z, delta_z, I):
    # spatial evolution 4th order RK
    # z is single value, delta_z is step
    k1 = TIE(z, I)
    k2 = TIE(z + delta_z / 2, I + k1 * delta_z / 2)
    k3 = TIE(z + delta_z / 2, I + k2 * delta_z / 2)
    k4 = TIE(z + delta_z, I + k3 * delta_z)
    return I + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)  # shape = (n_y, n_x)

```

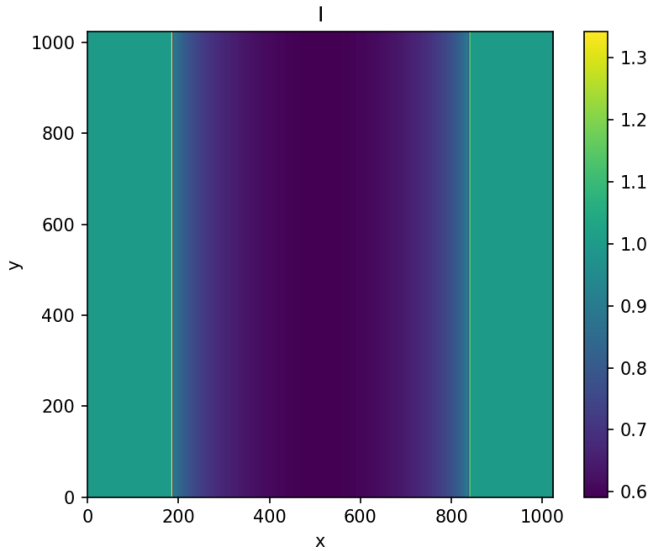
Testing

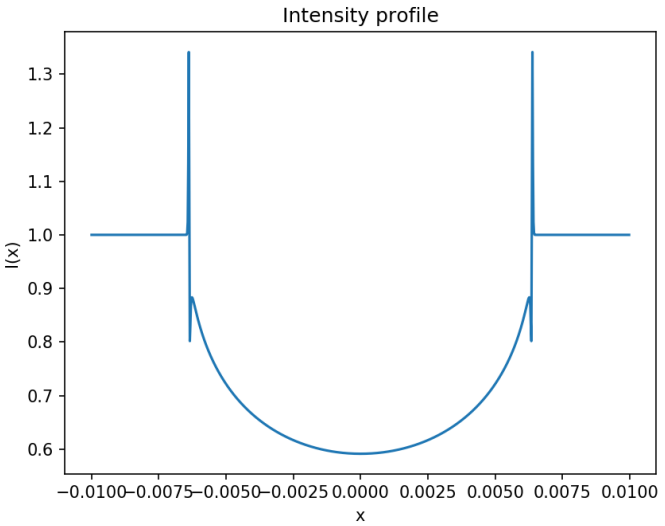
Using finite_diff with the non-Fourier TIE

x = np.linspace(-x_max, x_max, 1024, endpoint=False)
y= np.linspace(-10*mm, 10*mm, 1024, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)
σ = 0.01 * mm
Number of pixels used in blurring:
$\sigma \times (n_x/(x_{max} - x_{min})) = 0.512$

dx/dz = 1/4

```python
I_z = finite_diff(1 * m, I_0)
plot_I(I_z)
```



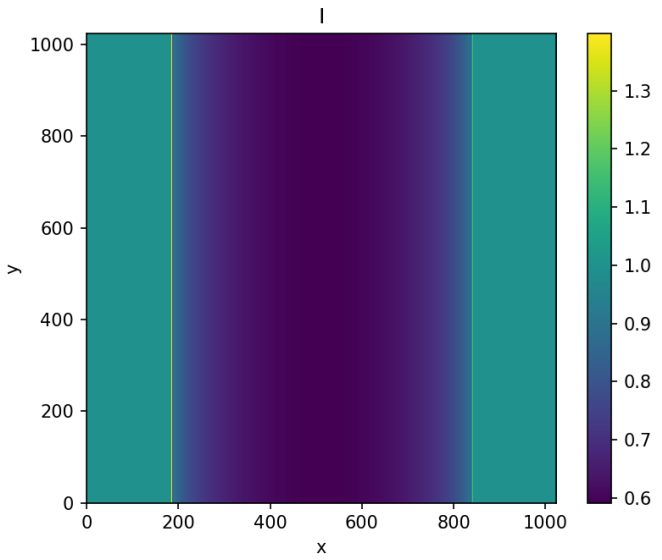


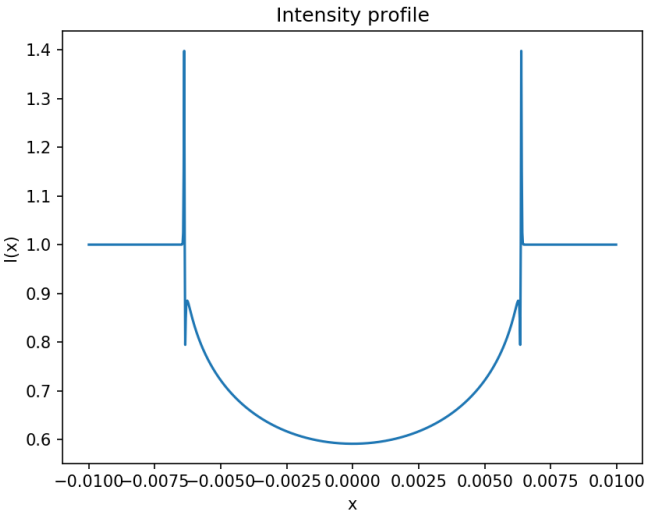
Using RK with the non-Fourier TIE

x = np.linspace(-10*mm, 10*mm, 1024, endpoint=False)
y= np.linspace(-10*mm, 10*mm, 1024, endpoint=False)
z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)

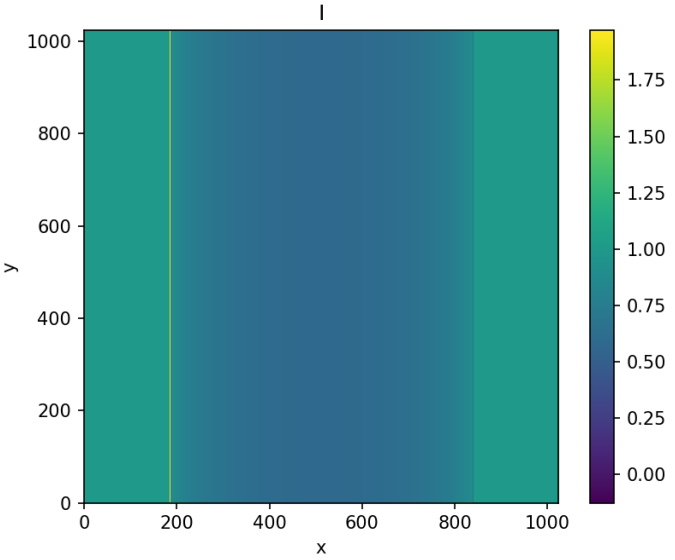
dx/dz = 1/4

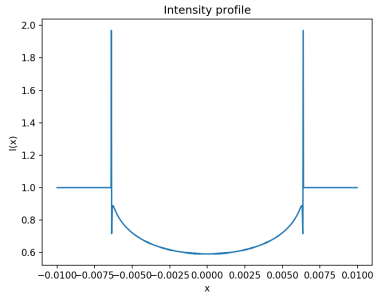
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min})) = 0.512$

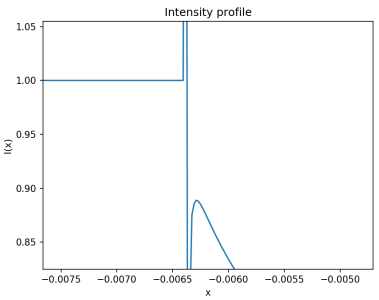


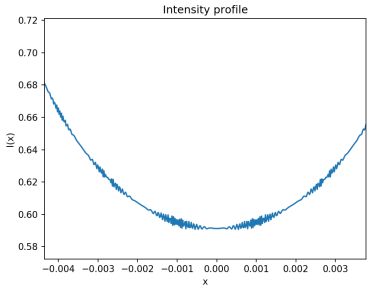


Using RK with the non-Fourier TIE
x = np.linspace(-x_max, x_max, **1024**, endpoint=False)
y= np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
z = np.linspace(-2 * R, 2 * R, **2 ** 12**, endpoint=False)
σ = **0.001 * mm**
Number of pixels used in blurring:
$\sigma \times (n_x/(x_{max} - x_{min})) = 0.0512$







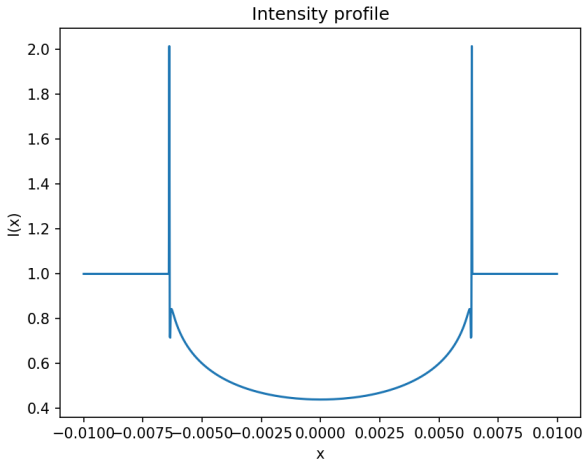


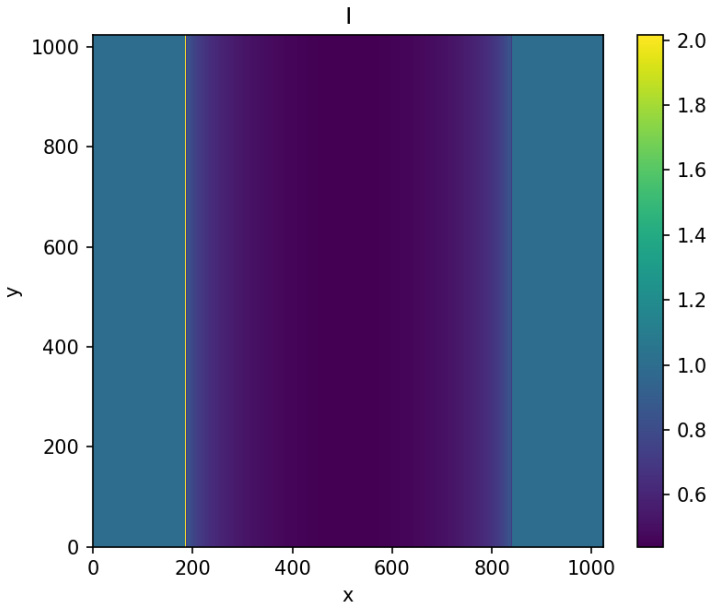
Even though it looks pretty good, I discovered that when the smoothing parameter is σ_x = 0.001 * mm the solution is still unstable!!

So I think I understand now why MK was right about using finite differences instead of FTs. The lower order polynomials used to fit the curves won't be vulnerable to overfitting, and subject to an undesirable effect like **Runge's phenomenon.**

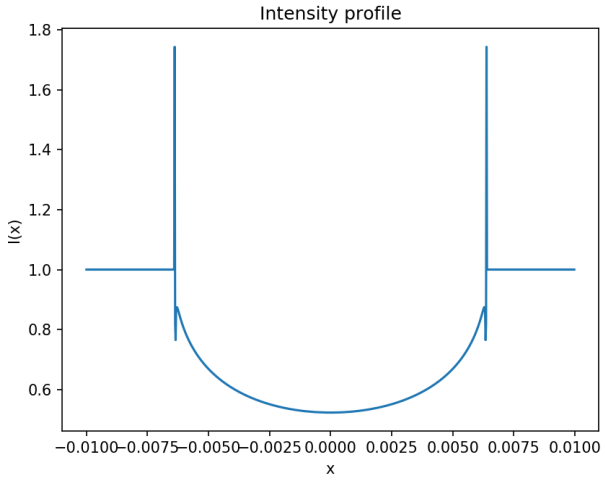
I asked Chris and he mentioned something called the **Gibbs phenomenon**: this is when the nth partial sum of a Fourier series (i.e. an infinite order polynomial) undergoes large oscillations near the regions with jump discontinuities. From Chris and the internet I also learnt about **Runge's phenomenon** which explains how sometimes using high degree polynomials over a set of interpolation points does not always improve accuracy.
I also want to test my code as is with the parameters in MK's simulation code:

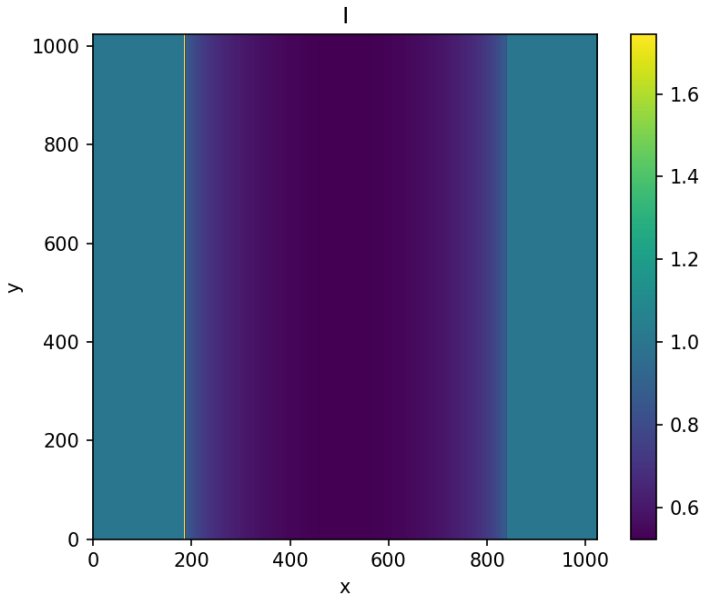
#Parameters as per energy_dispersion_Sim-1.py (MK's code)
energy1 = 3.5509e-15 * J # = 22.1629 * keV #- Ag k-alpha1
δ1 = 468.141 * nm
μ1 = 64.38436
λ = h * c / energy1
σ_x = 0.0027 * mm





energy2 = 3.996e-15 * J # = 24.942 * keV # - Ag k-beta1
δ1 = 369.763 *nm
μ1 = 50.9387
λ = h * c / energy2
σ_x = 0.0027 * mm





06/09/21

MK's latest comments

"Beautiful results - well done!
...it was interesting to see that the real-space solution was more stable (smooth) than the Fourier version. I think I've always done my TIE sims in real space, but the AS and Fresnel in Fourier space, which could partially explain why we see more ringing artefacts with those. Perhaps we should be doing everything in real space for the reasons you describe. You might be onto something important!
As for why your fringes look stronger than mine, I wonder if there is more smoothing happening in mine? Or maybe your Runge-Kutta approach is more effective than the finite difference approximation I use. You'd have to make all other parts of the code identical to check for the cause. Regardless, your code looks great - possibly better than mine/Florian's(!).
BTW, you should definitely keep all of these results for your final report, showing the work you did to perfect the simulations. It's not necessary to show all of the steps where there were mistakes, but these are genuine improvements with solid explanations that are worth sharing."

🙃

Current code status

```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.ndimage.filters import laplace
from physunits import m, cm, mm, nm, J, kg, s

plt.rcParams['figure.dpi'] = 150

# functions

def y_sigmoid(y):
    # smoothing out the edges of the cylinder in the y-direction
    σ_y = 0.004 * mm
    S = np.abs(1 / (1 + np.exp(-(y - height/2) / σ_y)) -
        (1 / (1 + np.exp(-(y + height/2) / σ_y))))
    return S # np.shape = (n_y, 1)


def δ(x, y, z):
    '''Refractive index: δ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    δ_array = δ0 * (1 / (1 + np.exp((r - R) / σ_x))) * y_sigmoid(y)
    return δ_array # np.shape(δ_array) = (n_y, n_x)


def μ(x, y, z):
    '''attenuation coefficient: μ0 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    μ_array = μ0 * (1 / (1 + np.exp((r - R) / σ_x))) * y_sigmoid(y)
    return μ_array # np.shape(μ_array) = (n_y, n_x)


def phase(x, y):
    # phase gain as a function of the cylinder's refractive index
    z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    Φ = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        Φ += -k0 * δ(x, y, z_value) * dz
    return Φ # np.shape(Φ) = (n_y, n_x)


def BLL(x, y):

    # TIE IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
    z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)
    dz = z[1] - z[0]
    # Euler's method
    F = np.zeros_like(x * y)
    for z_value in z:
        print(z_value)
        F += μ(x, y, z_value) * dz
    I = np.exp(- F) * I_initial
    return I # np.shape(I) = (n_y, n_x)


def gradΦ_laplacianΦ(Φ):
    dΦ_dx = np.gradient(Φ, delta_x, axis=1)
    dΦ_dy = np.gradient(Φ, delta_y, axis=0)
    lap_Φ = laplace(Φ / delta_x**2)
    return dΦ_dx, dΦ_dy, lap_Φ


def TIE(z, I):
    '''The intensity and phase evolution of a paraxial monochromatic
    scalar electromagnetic wave on propagation (2D)'''
    dI_dx = np.gradient(I, delta_x, axis=1)
    dI_dy = np.gradient(I, delta_y, axis=0)
    dI_dz = (-1 / k0) * (
        dI_dx * dΦ_dx +
        dI_dy * dΦ_dy +
        I * lap_Φ
        )
    return dI_dz  # np.shape(dI_dz) = (n_y, n_x)


def finite_diff(z, I):
    # first order finite differences
    I_z = I + z * TIE(z, I)
    return I_z


def Runge_Kutta(z, delta_z, I):
    # spatial evolution 4th order RK
    # z is single value, delta_z is step
    k1 = TIE(z, I)
    k2 = TIE(z + delta_z / 2, I + k1 * delta_z / 2)
    k3 = TIE(z + delta_z / 2, I + k2 * delta_z / 2)
    k4 = TIE(z + delta_z, I + k3 * delta_z)
    return I + (delta_z / 6) * (k1 + 2 * k2 + 2 * k3 + k4)  # shape = (n_y, n_x)
```

```
def propagation_loop(I_0):
    # RK Propagation loop parameters
    i = 0
    z = 0
    z_final = 1000 * mm
    delta_z = 1 * mm  # (n_z = 1000)

    I = I_0
    I_list = []
    while z < z_final:

        print(f"{i = }")

        # spatial evolution step
        I = Runge_Kutta(z, delta_z, I)
        if not i % 10:
            I_list.append(I)
        i += 1
        z += delta_z

    I_list = np.array(I_list)
    print(f"{np.shape(I_list) = }") #  np.shape(I_list) = (n_z / 10, n_y, n_x)
    # np.save(f'I_list.npy', I_list)
    return I_list

def plot_I(I):
    # PLOT Phase contrast I in x, y
    plt.imshow(I, origin='lower')
    plt.colorbar()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("I")
    plt.show()

    # PLOT I vs x (a single slice)
    plt.plot(x, I[np.int(n_y / 2),:])
    plt.xlabel("x")
    plt.ylabel("I(x)")
    plt.title("Intensity profile")
    plt.show()

def globals():

    # constants
    h = 6.62607004e-34 * m**2 * kg / s
    c = 299792458 * m / s

    # x-array parameters
```



```
    n = 1024

    n_x = n
    x_max = 10 * mm
    x = np.linspace(-x_max, x_max, n_x, endpoint=False)
    delta_x = x[1] - x[0]

    # y-array parameters
    n_y = n
    y_max = 10 * mm
    y = np.linspace(-y max, y max, n_y, endpoint=False)#.reshape(n_y, 1)
    delta_y = y[1] - y[0]
    y = y.reshape(n_y, 1)

    # X-ray beam parameters
    E = 3.845e-15 * J # (Beltran et al. 2010)
    λ = h * c / E
    k0 = 2 * np.pi / λ  # x-rays wavenumber

    # # refraction and attenuation coefficients
    δ0 = 462.8 * nm # (Beltran et al. 2010)
    μ0 = 41.2 # per meter # (Beltran et al. 2010)

    # Cylinder parameters
    D = 12.75 * mm
    R = D / 2
    z_c = 0 * mm
    x_c = 0 * mm
    height = 20 * mm
    σ_x = 0.01 * mm

    return x, y, n_x, n_y, delta_x, delta_y, k0, R, z_c, x_c, δ0, μ0, height, σ_x
```

Week 7 (06/09/21 –12/09/21)

Aims:

1. Create simulations of phase contrast imaging through different densities and materials

Tasks:

1. Meetings on **Monday 06/09** and **Wednesday 08/09**
    a. ~~Group: **2 pm**~~
    b. ~~one-on-one: **11 am**~~
2. ~~Fix sim or swap to Florian's code~~
3. **Read PyQt GUI book**
4. **Plan progress report wk 9**

06/09/21

2 cylinders

MK's earlier comments:
We want to see:

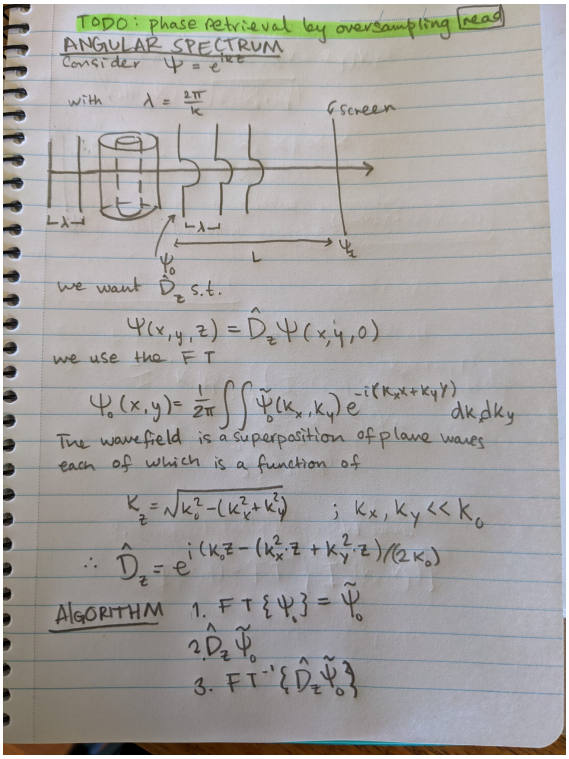
a) whether density changes alone will give phase contrast and (as I think they will), how small the fringes will be;
b) how the fringes change when we start changing the spectrum.
Of course for part b) we'll really need to use the angular spectrum or Fresnel codes, so you'll need those libraries working.

Adapting my code to match the refractive indices and attenuation coefficients of two concentric cylinders

```python
def δ(x, y, z, δ1, δ2):
    '''Refractive index: δ1 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    δ_array = (δ1 * (1 / (1 + np.exp((r - R) / σ_x))) + (δ2 - δ1) * (1 / (1 + np.exp((r - R) / σ_x)))) * y_sigmoid(y)
    return δ_array # np.shape(δ_array) = (n_y, n_x)


def μ(x, y, z, μ1, μ2):
    '''attenuation coefficient: μ1 within the cylinder
    decreasing to zero at the edges Sigmoid inspired:'''
    r = np.sqrt((x - x_c) ** 2 + (z - z_c) ** 2)
    μ_array = (μ1 * (1 / (1 + np.exp((r - R) / σ_x))) + (μ2 - μ1) * (1 / (1 + np.exp((r - R) / σ_x)))) * y_sigmoid(y)
    return μ_array # np.shape(μ_array) = (n_y, n_x)
```

Angular Spectrum Formulation

(Als-Nielsen and McMorrow 324)



```python
def Angular_spectrum(z, field):
    """Wave propagation using the angular spectrum method.
    Code follows Als-Nielsen, Elements of Modern X-ray Physics, p.324"""
    D_operator = np.exp(1j * k0 * z) * np.exp(-1j * (kx**2 * z + ky**2 * z) / (2 * k0))
    FT_field = fft2(field)
    return ifft2(D_operator * FT_field)
```

07/09/21

Testing for 1 cylinder only

Changes TODO:

1. δ(x, y, z, δ1)**:#**, δ2):
    a. δ_array = (δ1 * (1 / (1 + np.exp((r - R) / σ_x))) **)#** + (δ2 - δ1) * (1 / (1 + np.exp((r - R) / σ_x)))) * y_sigmoid(y)
2. μ(x, y, z, μ1)**:#**, μ2):
    a. μ_array = (μ1 * (1 / (1 + np.exp((r - R) / σ_x))) **)#** + (μ2 - μ1) * (1 / (1 + np.exp((r - R) / σ_x)))) * y_sigmoid(y)
3. Φ = -k0 * δ(x, y, z_value, δ1, δ2)
4. F = μ(x, y, z_value, μ1, μ2)
5. return x, y, n_x, n_y, delta_x, delta_y, k0, kx, ky, R, R2, z_c, x_c, δ1, μ1, σ_x, height **# add δ2, μ2,**
6. x, y, n_x, n_y, delta_x, delta_y, k0, kx, ky, R, R2, z_c, x_c, δ1, μ1, σ_x, height = globals() **# add δ2, μ2,**

Current testing parameters

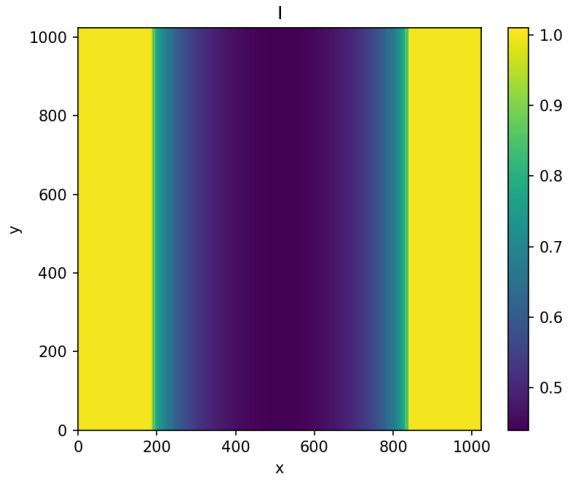
For a single cylinder:

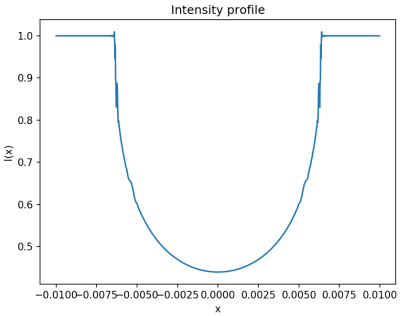
energy1 = 3.5509e-15 * J # = 22.1629 * keV #- Ag k-alpha1
δ1 = 468.141 * nm
μ1 = 64.38436
λ = h * c / energy1
k0 = 2 * np.pi / λ # x-rays wavenumber
kx = 2 * np.pi * np.fft.fftfreq(n_x, delta_x)
ky = 2 * np.pi * np.fft.fftfreq(n_y, delta_y).reshape(n_y, 1)
σ_x = 0.0027 * mm

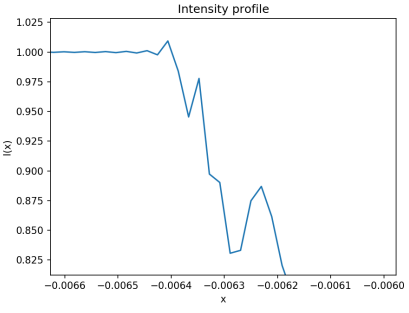
```python
    # # ICS
    I_initial = np.ones_like(x * y)
    I_0 = BLL(x, y)
    Φ = phase(x, y)

    Ψ_0 = np.sqrt(I_0) * np.exp(1j*Φ)
```

output:

```python
    z_final = 1 * m
    Ψ = Angular_spectrum(z_final, Ψ_0)
    I = np.abs(Ψ**2)

    plots_I(I)
```

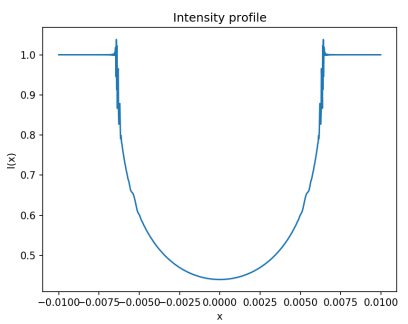


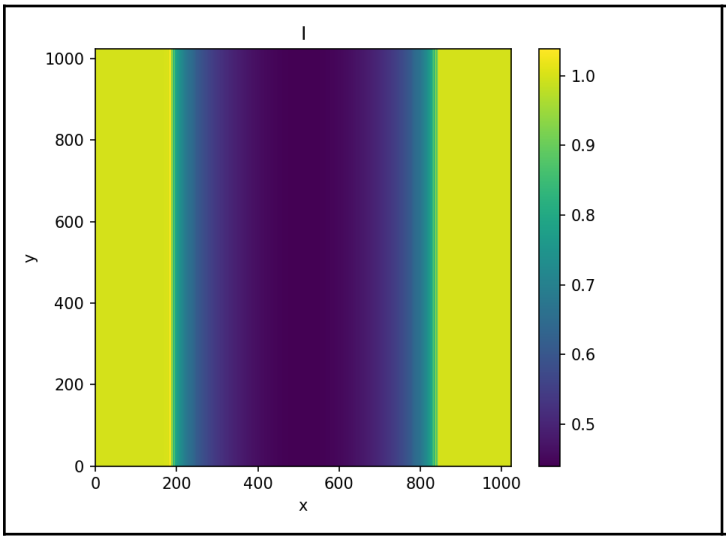


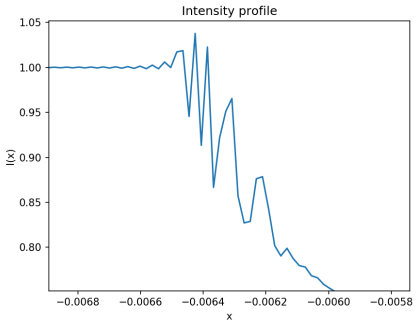


Copying my previous work

x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
y= np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
z = np.linspace(-2 * R, 2 * R, **65536**, endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x / (x_{max} - x_{min})) = 0.5$

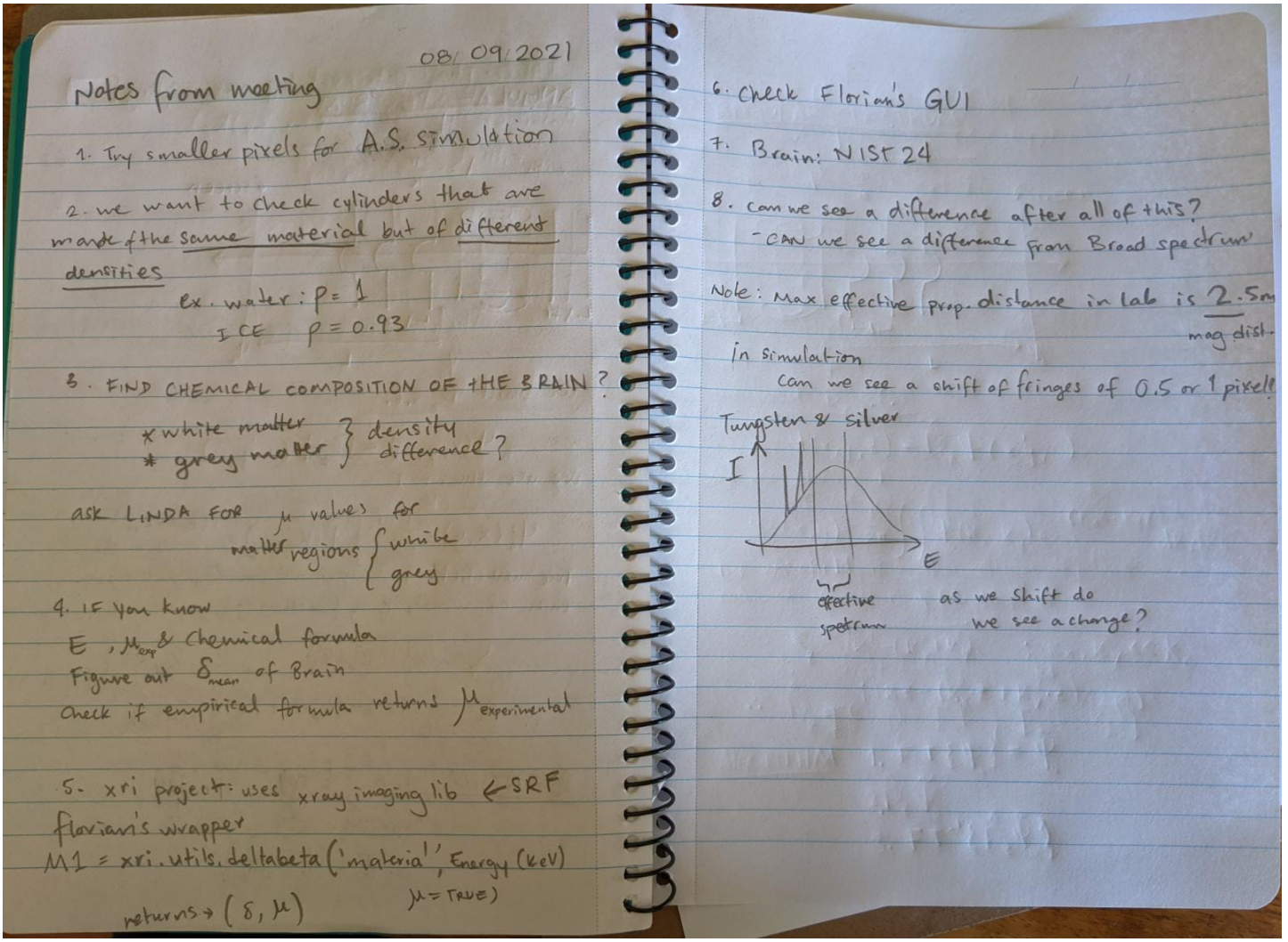






08/09/21

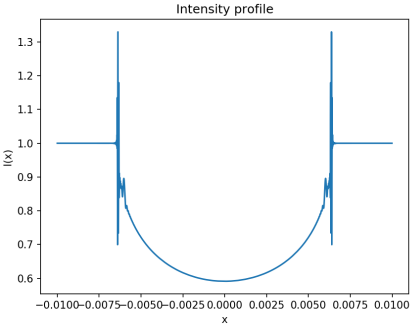
MEETING DAY!

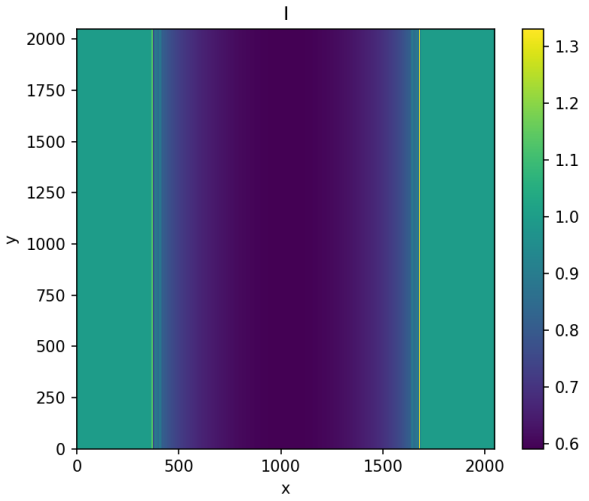


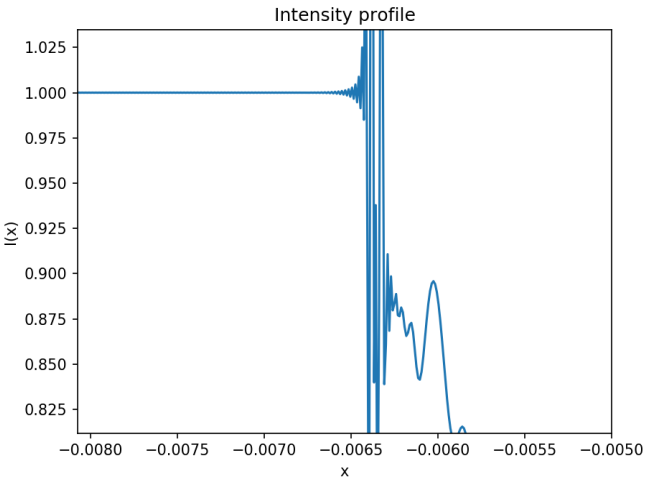
Trying out **smaller pixels**

x = np.linspace(-10*mm, 10*mm, **2048,** endpoint=False)
y= np.linspace(-10*mm, 10*mm, **2048,** endpoint=False)
z = np.linspace(-2 * R, 2 * R, **65536**, endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 1.024

dx/dz = 1/32







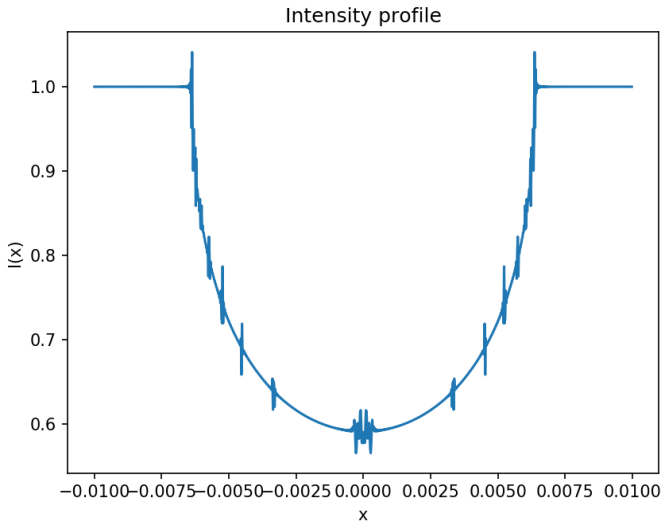
From now on I won't bother calculating y

My integration step to find the ICs is too rudimentary. I'll look into something better to integrate faster
I opted for **scipy.integrate.quad**(*func*, *a*, *b*)
Integrate func from *a* to *b* (possibly infinite interval) using a technique from the Fortran library QUADPACK.

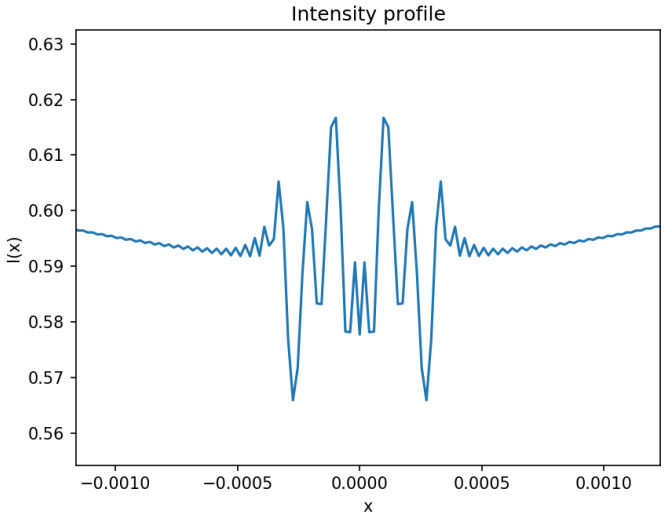
```
36  def phase(x, y):
37      # phase gain as a function of the cylinder's refractive index
38      Φ = np.zeros_like(x * y)
39      for i, y_val in enumerate(y):
40          for j, x_val in enumerate(x):
41              Φ[i, j] = -k0 * integrate.quad(lambda z: δ(x_val, y_val, z, δ1),-2 * R, 2 * R)[0] #, δ2)
42              print(f"{Φ[i, j]}")
43      return Φ # np.shape(Φ) = (n_y, n_x)
44
45
46  def BLL(x, y):
47      # TIE IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
48      F = np.zeros_like(x * y)
49      for i, y_val in enumerate(y):
50          for j, x_val in enumerate(x):
51              F[i, j] = integrate.quad(lambda z: μ(x_val, y_val, z, μ1),-2 * R, 2 * R)[0] #, μ2)
52              print(f"{F[i, j]}")
53      I = np.exp(- F) * I_initial
54      return I # np.shape(I) = (n_y, n_x)
```

Testing

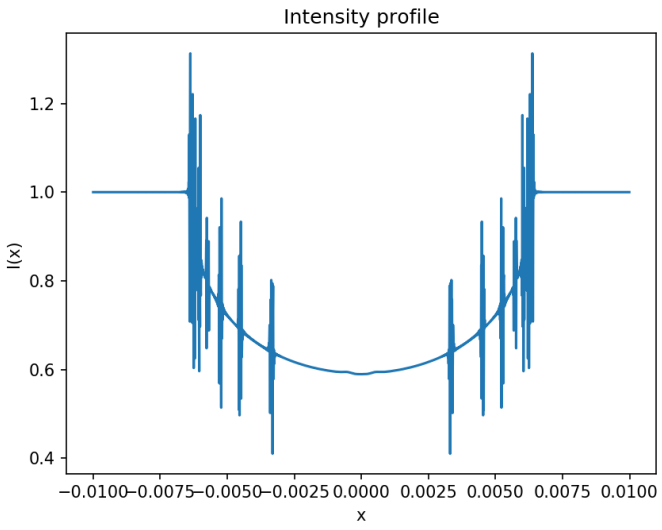
Propagated distance: z_final = 1 * m
x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
y= np.linspace(-10*mm, 10*mm, **20,** endpoint=False)
z = integrate.quad decides best!
σ = **0.0027 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.14

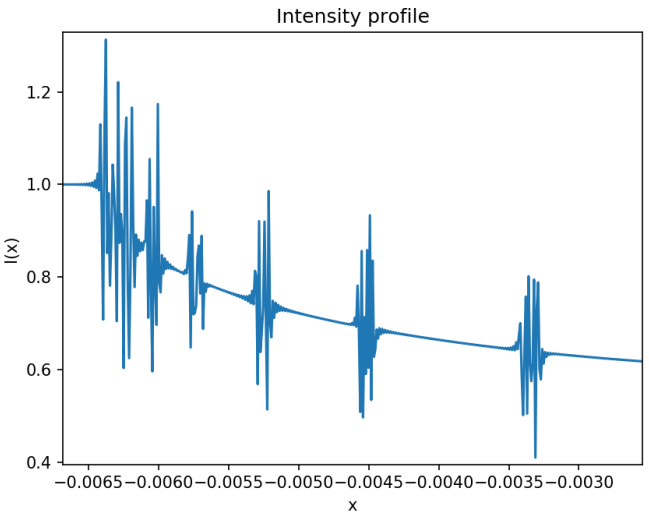


There is periodicity in the wiggles!

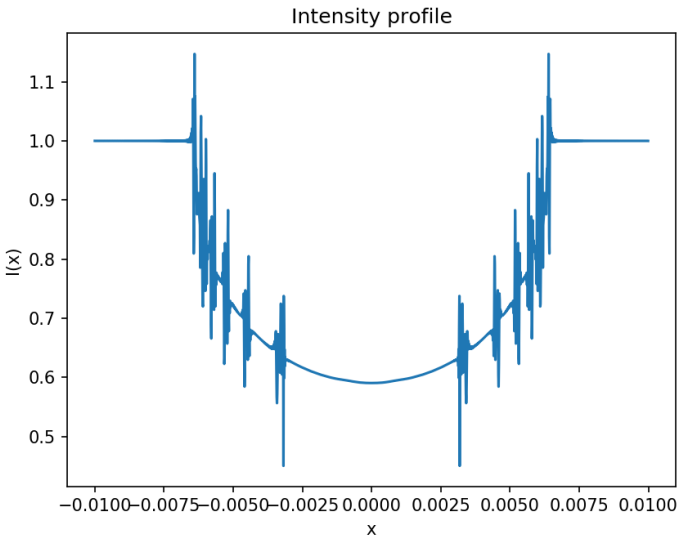


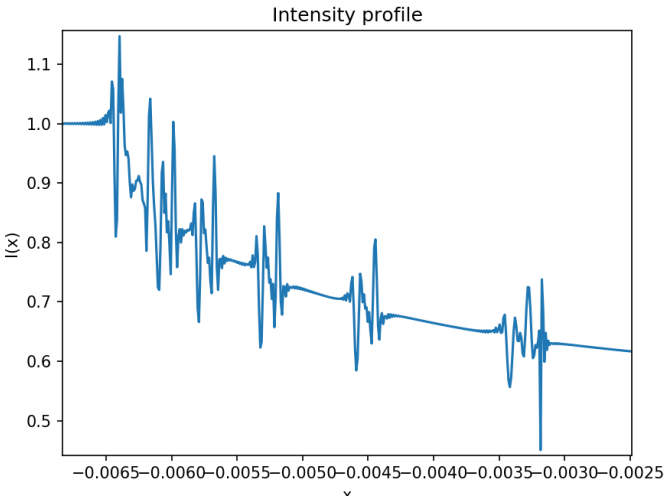
z_final = 1 * m
x = np.linspace(-10*mm, 10*mm, **2048,** endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 1.02





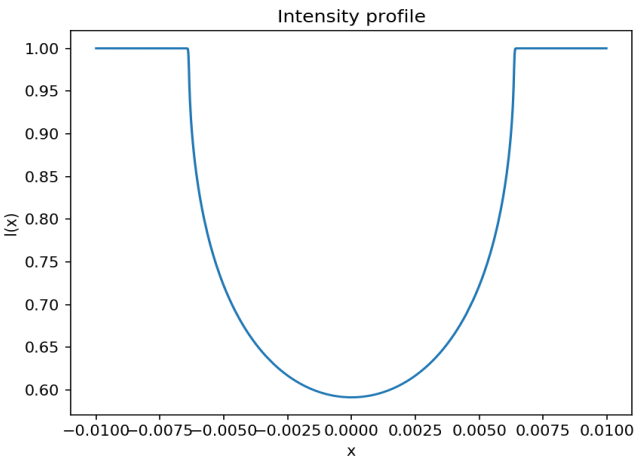
z_final = 1 * m
x = np.linspace(-10*mm, 10*mm, **2048,** endpoint=False)
σ = **0.03 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 3.07



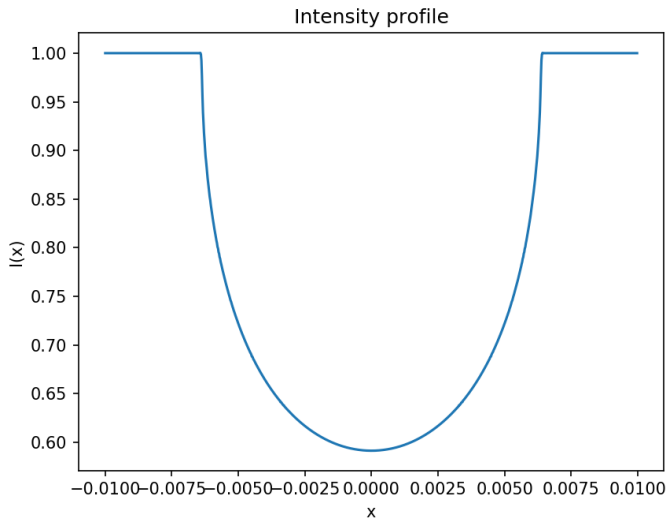


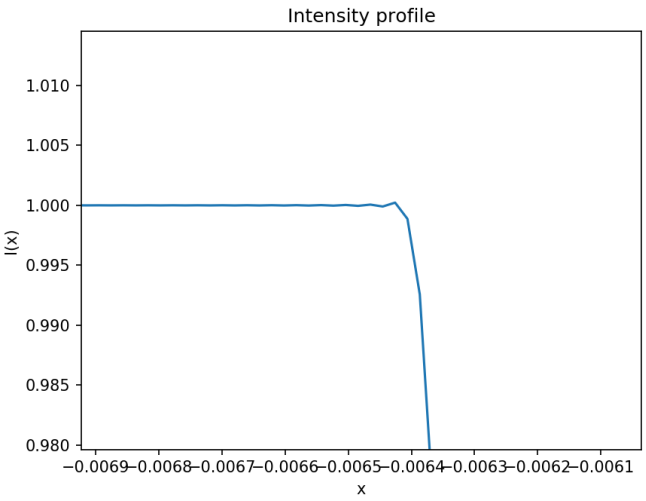
WHY are things so explodey?

What does the profile look like right after passing through the cylinder?
z_final = 1 * mm
x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.51

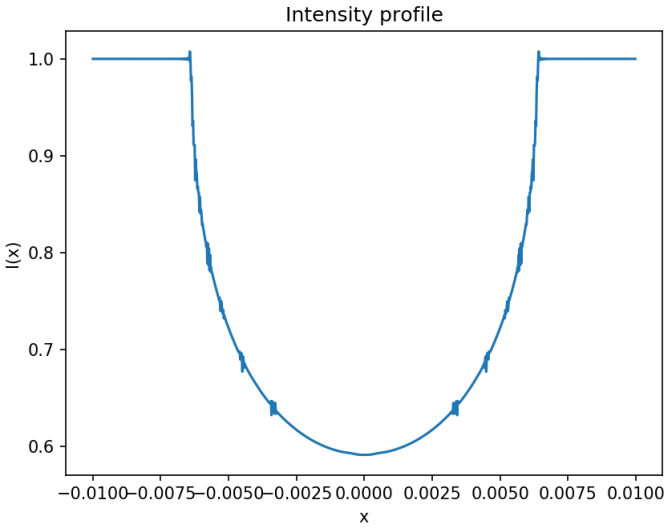


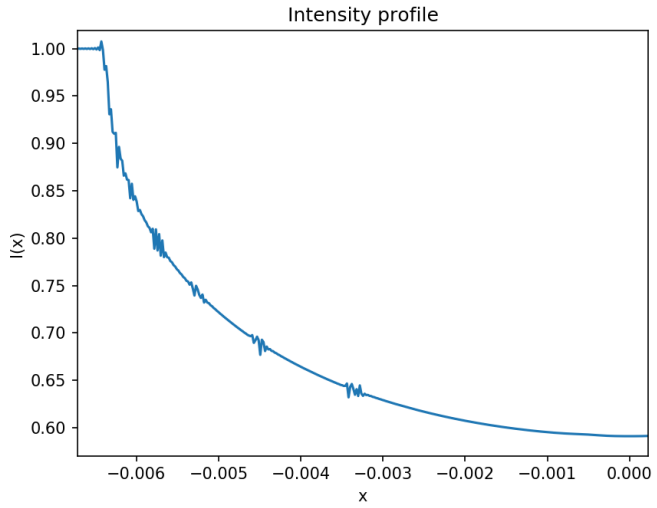
z_final = **10 * mm**
x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.51





z_final = **20 * cm**
x = np.linspace(-10*mm, 10*mm, **1024,** endpoint=False)
σ = **0.01 * mm**
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min})) = 0.51$



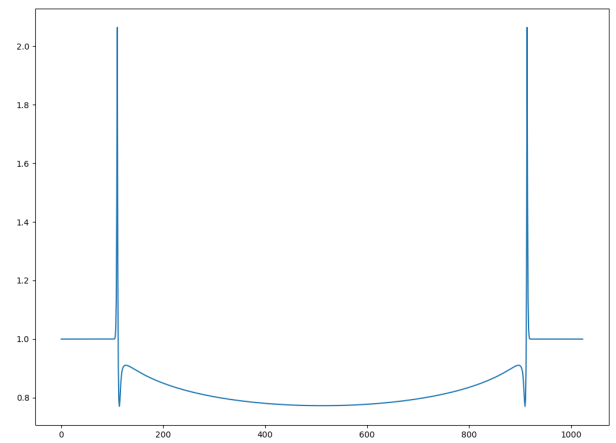


09/09/2021

Comparing to MK's code

This is the intensity profile from **Energy_Dispersion_Sim-1.py**

```
# Material = water
energy1=22.1629 #keV - Ag k-alpha1
delta1=4.68141e-07
k1=xri.utils._conversions.energy2k(energy1)
mu1=64.38436
beta1=mu1/(2.*k1)
xsize=1024
ysize=xsize
pxl=5E-6
z=1. # Prop distance, m

# function call
Prop1=xri.sim.propAS(delta1*T, beta1*T, energy1, z, pxl,
supersample=3)
plt.figure()
Line1=Prop1[900]
plt.plot(Line1, label='Prop_E1')
plt.show()
```

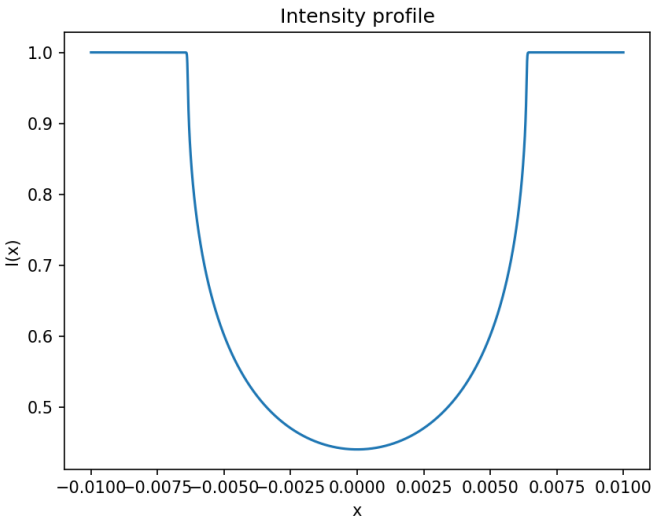


Pixels in MK's sim are set to one order of magnitude smaller than mine!
pxl = $\frac{20\,mm}{1024}$ =**2E-5**
Here I've increased the number of x points --> n_x = 214 (~ 10 times smaller pixels)**
pxl = $\frac{20\,mm}{2^{14}}$ =**1E-6**

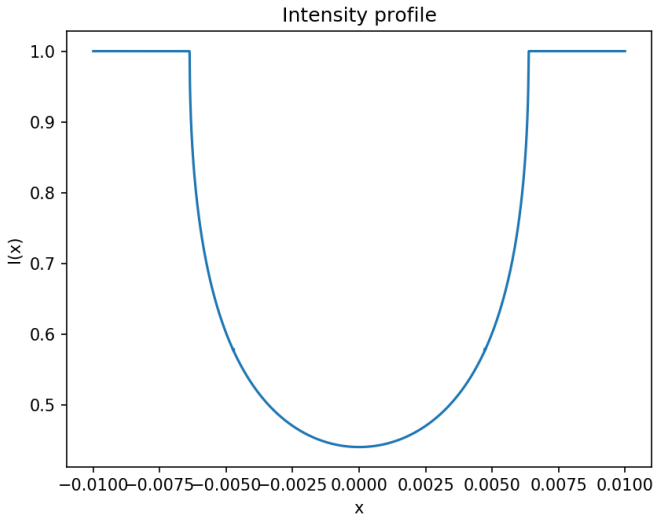
```
# Material = water
```

E=22.1629 * keV # Ag k-alpha1
δ1=4.68141e-07
μ1=64.38436

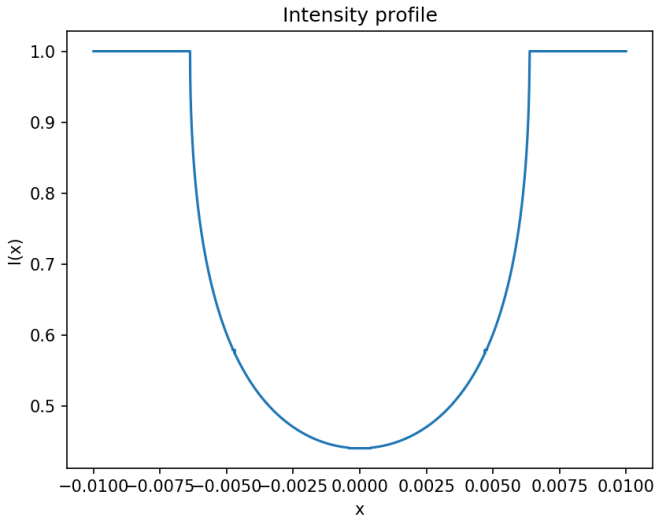
n_x = **2 ** 14**
x_max = 10 * mm
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
delta_x = x[1] - x[0]
n_y = 20 (for speed)
y_max = 10 * mm
y = np.linspace(-y_max, y_max, n_y, endpoint=False)#.reshape(n_y, 1)
y = y.reshape(n_y, 1)
pxl= $\frac{20\ mm}{2^{14}}$ =1E-6
# Blurring
σ_x = 0.01 * mm ← possibly too much
Number of pixels used in blurring:
$\sigma \times (n_x/(x_{max} - x_{min}))$ = 8.192
z_final = 1 * m
# function call
Ψ = Angular_spectrum(z_final, Ψ_0)
I = np.abs(Ψ**2)
plots_I(I)

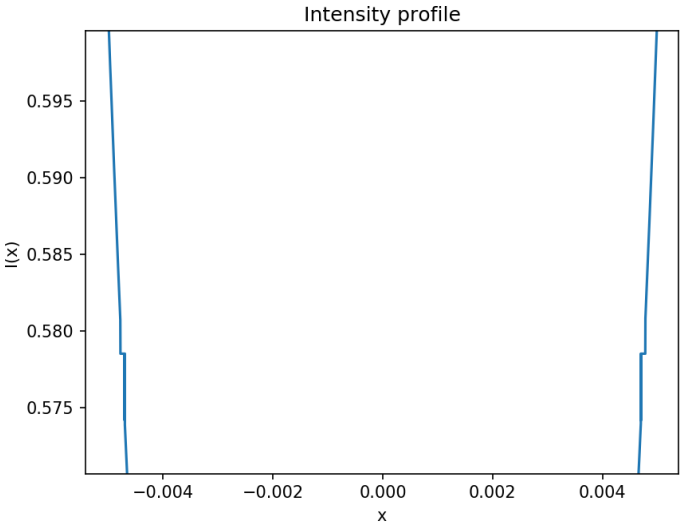


Blurring
σ_x = 0.001 * mm
Number of pixels used in blurring:
$\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.8192



Blurring
σ_x = 0.0001 * mm
Number of pixels used in blurring:
$\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.0819





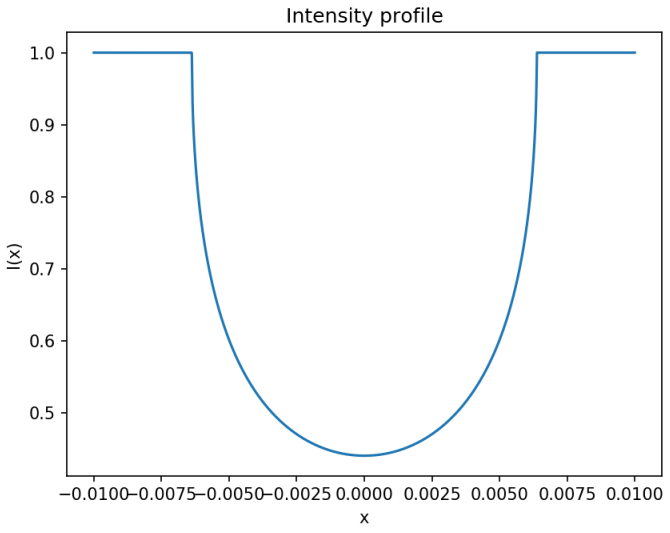
Context switching

Since I haven't been able to figure out how to decrease the periodic instability yet, while at the same time obtaining some phase contrast with my AS propagation formulation I decided to change my methodology to match Florian and MK's code more closely.

```
import xri
from scipy.ndimage import gaussian_filter
```

```
# functions I removed:
y_sigmoid(y)
δ(x, y, z, δ1):#, δ2)
μ(x, y, z, μ1):#, μ2)
# functions I added:
def thicc(x, y):
        # # Create cylindrical object projected thickness
        T = np.zeros_like(x * y)
        T[0:20,0:100] = 1
        T= 2 * np.sqrt(R ** 2 - x ** 2)
        T = np.nan_to_num(T)
        T = gaussian_filter(T, sigma=2)
        ones_y=np.ones_like(y)
        # # Expand 1D to 2D with outer product
        T = np.outer(ones_y, T)
        # im = plt.imshow(T)
        return T

# functions I changed
phase(x, y)
def phase(x, y, δ):
        # phase gain as a function of the cylinder's refractive index
        Φ = - k0 * δ * thicc(x, y)
        return Φ # np.shape(Φ) = (n_y, n_x)
BLL(x, y)
def BLL(x, y, μ):
        # IC of the intensity (z = z_0) a function of the cylinder's attenuation coefficient
        I = np.exp(- μ * thicc(x, y)) * I_initial
        return I # np.shape(I) = (n_y, n_x)
FUNCTION CALLS
        x, y, n_x, n_y, delta_x, delta_y, k0, kx, ky, R, z_c, x_c, δ1, μ1, height = globals()
        # # ICS
        I_initial = np.ones_like(x * y)
        I_0 = BLL(x, y, μ1)
        Φ = phase(x, y, δ1)
        Ψ_0 = np.sqrt(I_0) * np.exp(1j*Φ)

        z_final = 1 * m
        Ψ = Angular_spectrum(z_final, Ψ_0)
        I = np.abs(Ψ**2)
        plots_I(I)
```

```
CURRENT GLOBALS
h = 6.62607004e-34 * m**2 * kg / s
c = 299792458 * m / s
# # x-array parameters
n = 1024
n_x = n
x_max = 10 * mm
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
delta_x = x[1] - x[0]
# # y-array parameters
n_y = 20 # n
y_max = 10 * mm
y = np.linspace(-y_max, y_max, n_y,   endpoint=False)#.reshape(n_y, 1)
delta_y = y[1] - y[0]
y = y.reshape(n_y, 1)
# # Parameters from Energy_Dispersion_Sim-1.py
# # Material = water
E = 22.1629 * keV # Ag k-alpha1
δ1 = 4.68141e-07
μ1 = 64.38436
λ = h * c / E
# wave number
k0 = 2 * np.pi / λ  # x-rays wavenumber
# For Fourier space
kx = 2 * np.pi * np.fft.fftfreq(n_x, delta_x)
ky = 2 * np.pi * np.fft.fftfreq(n_y, delta_y).reshape(n_y, 1)
# Cylinder parameters
D = 12.75 * mm
R = D / 2
z_c = 0 * mm
x_c = 0 * mm
height = 20 * mm
```



Wow this is so much faster!

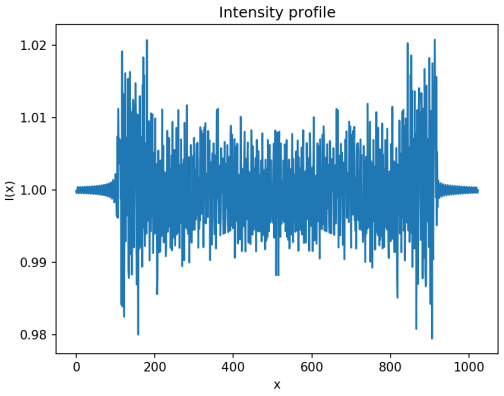
But this profile looks the same as with my previous code!

I'm clearly doing something wrong! Since nothing is happening after propagation.

10/09/21

Morphing into MK/Florians's code

I must test inputs and see where mine differ from MK's

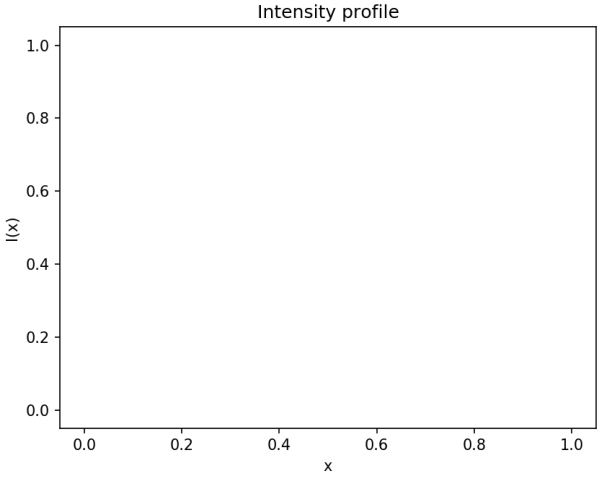
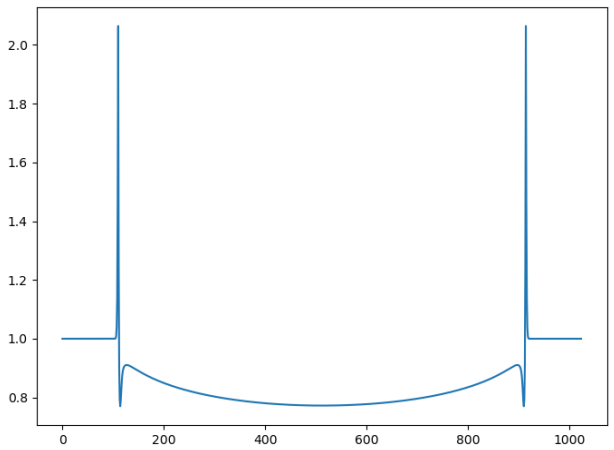
Mine	MK's
Printing the first element of the x array: x[0] =-0.01	x[0] = -0.00256
Printing the last element of the x array: x[-1] = 0.00998046875	x[-1] = 0.0025550000000000004
My solution: n_x = 1024 x_max = (n_x / 2) * 5 * um x = np.linspace(-x_max, x_max, n_x, endpoint=False)	**x[0] = -0.0025599999999999998** **x[-1] = 0.002555** Hopefully this is good enough?
Output:  	HAHAHA wow.

```
[ana:code]$ python3 2_cylinders.py
2_cylinders.py:18: RuntimeWarning: invalid value encountered in sqrt
  T = 2 * np.sqrt(R ** 2 - x ** 2)

delta_x =4.999999999999796e-06

E1 =22162.9

k0=7.010189049709009e+29

R = 0.002

T = array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

np. all((T == 0)) = False

δ1 = 4.68141e-07

μ1 = 64.38436
```

```
[ana:code]$ python3 Energy_Dispersion_Sim-1.py
Energy_Dispersion_Sim-1.py:47: RuntimeWarning: invalid value encountered in sqrt
  T=2.*np.sqrt(Radius**2.-x**2.)

x[1] - x[0]=4.999999999999796e-06

energy1=22.1629

k1=112315609089.10257

Radius = 0.002

T = array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

np. all((T == 0)) = False

delta1 = 4.68141e-07

mu1 = 64.38436
```

WOW, my energy value is wrong! By a factor of 1000!!

I think there might be a bug in the **physunits** library because keV might be 1000 larger than it should?

I converted to Joules manually: E1 =3.5508879e-15 k0=112315606140.55118  Boo! Nothing happens...	energy1=22.1629 k1=112315609089.10257  The difference between our k values >>> k1 - k0 = 2948.5513916015625
I calculate k0 by h = 6.62607004e-34 * m**2 * kg / s c = 299792458 * m / s	How does Florian calculate the k1 value? k = energy2k(energy)

```
E1 = 3.5508879e-15 * J
λ = h * c / E1

# wave number
k0 = 2 * np.pi / λ  # x-rays wavenumber
```

```
def energy2k(energy):
    return 2*np.pi/energy2wl(energy)
def energy2wl(energy):
    return const.h/const.eV * const.c/(energy*1000)
```

Solution:
import scipy.constants as const
h = const.h
c = const.c

Previous and current output:

k0=112315606140.55118
k0=112315604275.98947

k1=112315609089.10257

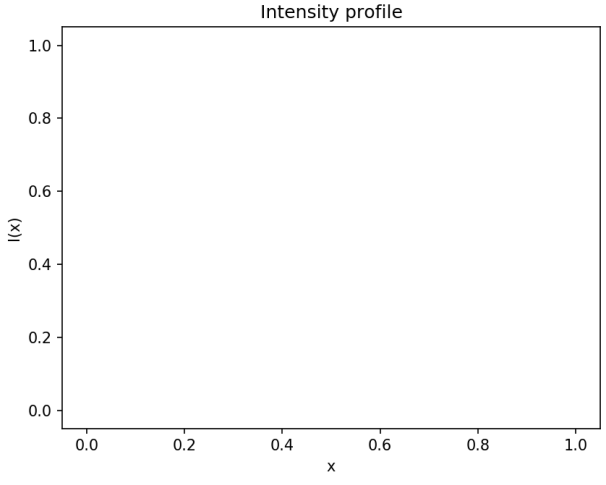
The difference between our k values
>>> k1 - k0 = 4813.113098144531
Now our results differ more? So strange...

```
E1 =3.5508879e-15

k0=112315604275.98947

beta1 = 2.866225063517907e-10

Φ = array([[-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       ...,
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.]])

np. all((Φ == 0)) = False

I_0 = array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]])


np. all((I_0 == 1)) = False
```

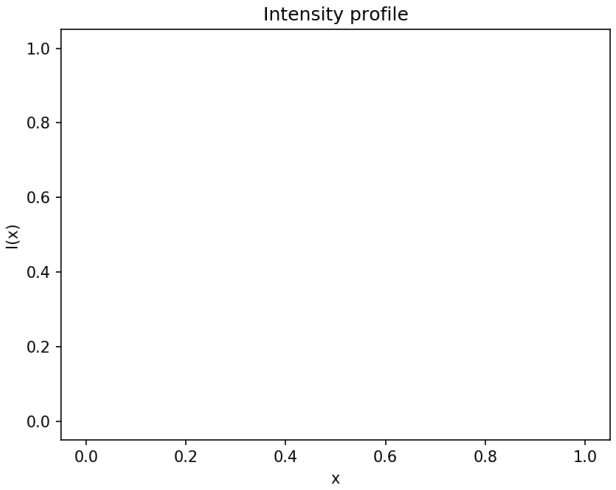
```
energy1=22.1629

k1=112315609089.10257

beta1 = 2.8662249406902294e-10

phase1 = array([[-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       ...,
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.],
       [-0., -0., -0., ..., -0., -0., -0.]])

np. all((phase1 == 0)) = False

Int1 = array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]])


np. all((Int1 == 1)) = False
```

It looks like the main obvious difference in inputs is the k value

Prop1 = xri.sim.propAS(δ1*T, beta1*T, E1, z_final, delta_x, supersample=3)
I = Prop1[0]
plots_I(I)

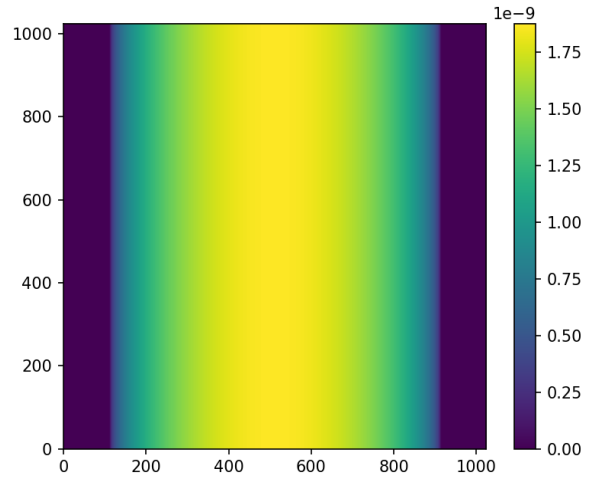


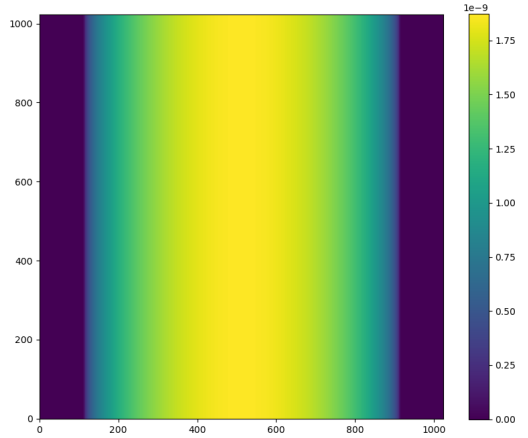
Prop1 = xri.sim.propAS(δ1*T, beta1*T, E1, z_final, delta_x, supersample=3)
I = Prop1[-1]
plots_I(I)



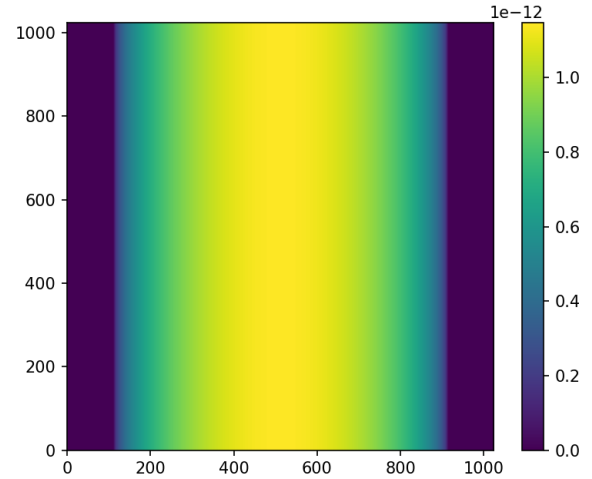
OK so I'm likely screwing up the function call ...
Prop1 = xri.sim.propAS(δ1*T, beta1*T, E1, z_final, delta_x, supersample=3)
Let's check each input individually

Mine	MK's
δ1*T	**delta1*T**

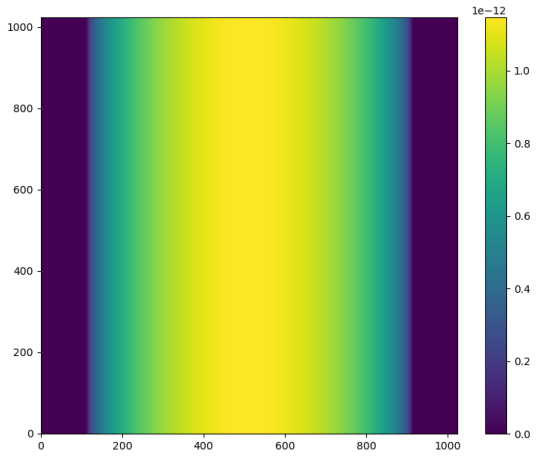




beta1*T



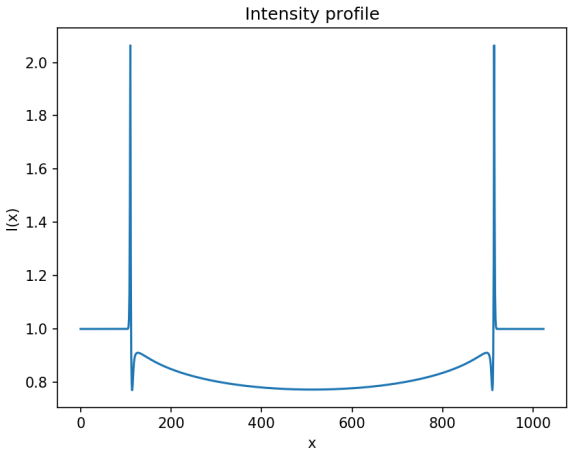
beta1*T



E1
My input is in J
DUH!
I'll change my E1
E1 = 22.1629 # keV # Ag k-alpha1

energy1
MK's input is in keV
From Florian's documentation:
energy : float or array
Energy in [keV]

Prop1 =
xri.sim.propAS(δ1*T, beta1*T, E1, z_final, delta_x, supersample=3)
I = Prop1[900]
plots_I(I)



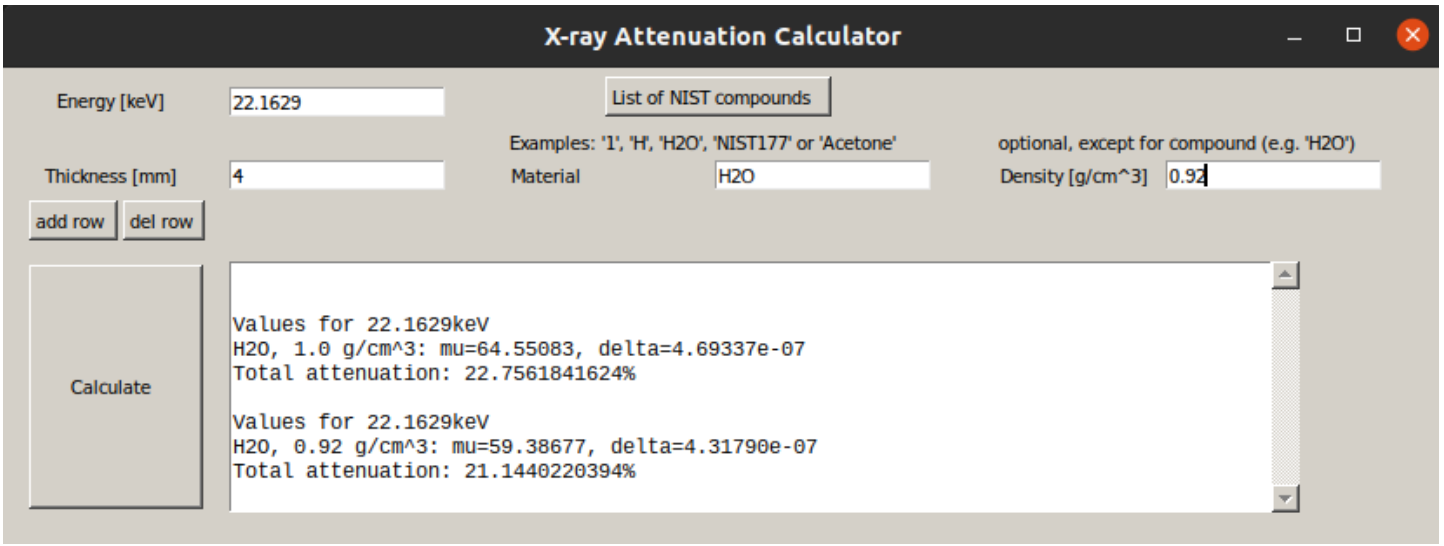
YAY! SUCCESS

11/09/21

Water & Ice cylinders

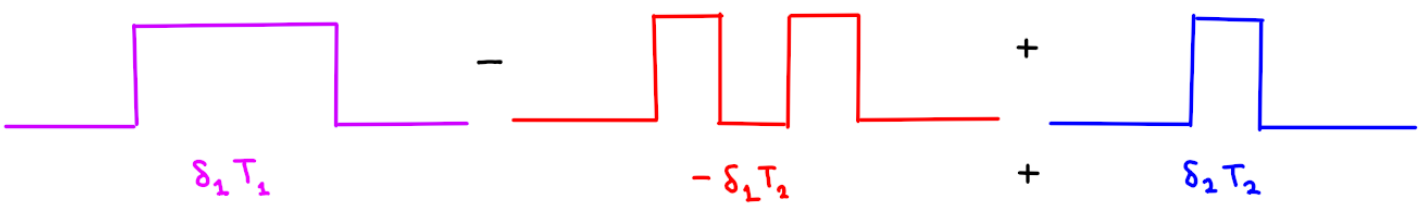
I am using Florian's calculator to find my desired simulation parameters. This calculator allowed me to simply input the material density and an energy value to try in my simulation.

Linear Attenuation coefficients and different densities

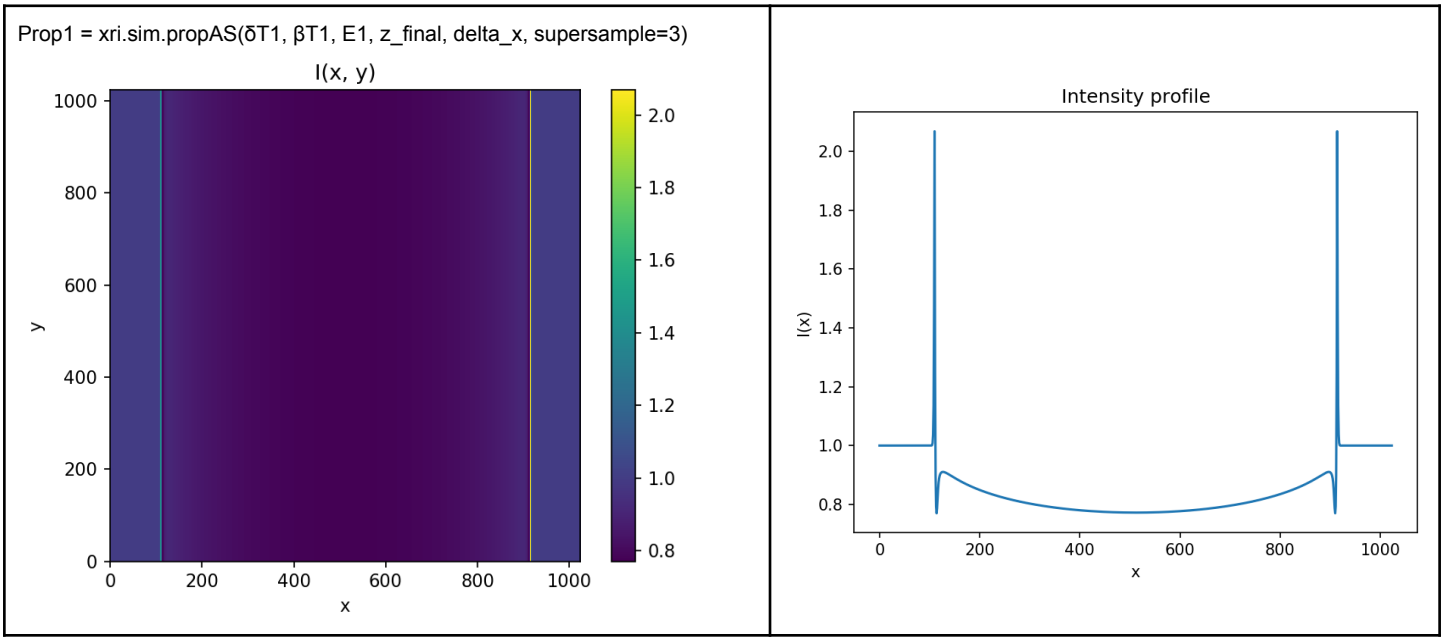



```
57      # Parameters from X-ray attenuation calculator
58      E1 = 22.1629 # keV # Ag k-alpha1
59      λ = h * c / (E1 * 1000 * const.eV)
60      k1 = 2 * np.pi / λ  # x-rays wavenumber
61      # Material = water, density = 1 g/cm**3
62      δ1 = 469.337 * nm
63      μ1 = 64.55083 # per cm
64      β1 = μ1 / (2 * k1)
65      # Material = ice, density = 0.92 g/cm**3
66      δ2 = 431.790 * nm
67      μ2 = 59.38677 # per cm
68      β2 = μ2 / (2 * k1)
```

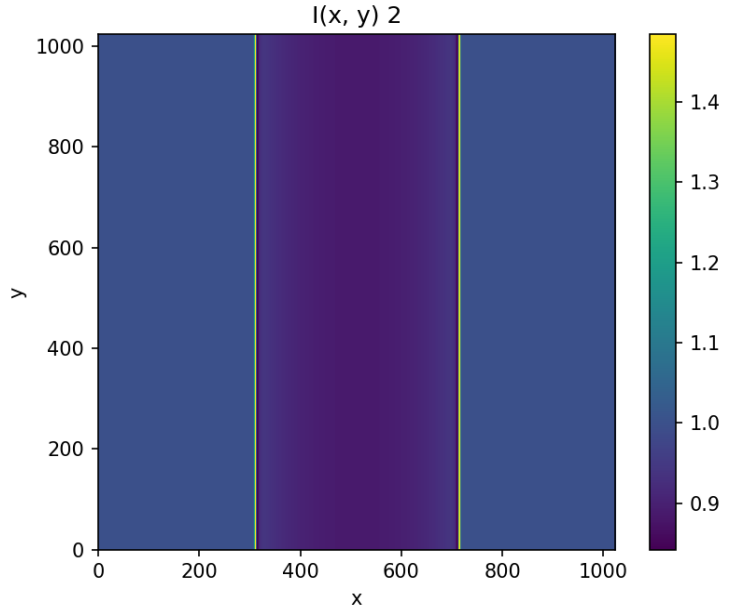
Whoops the units of mu are supposed to be in m not cm :P
To avoid double counting given that one cylinder is inside the other I thought of this setup

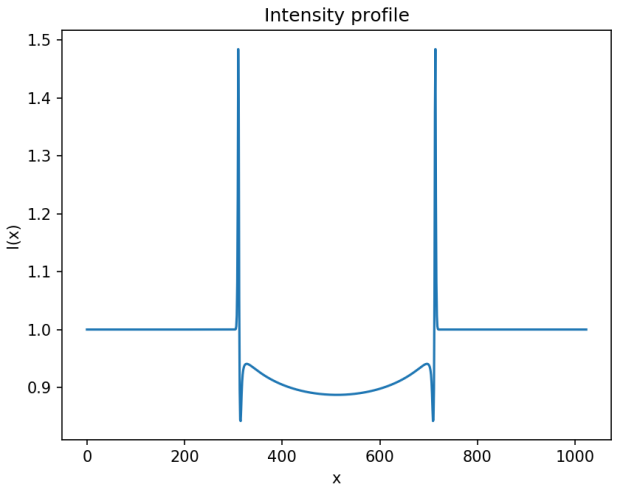



```
T1 = thicc(x, y, R1)
T2 = thicc(x, y, R2)
δT1 = δ1 * T1
βT1 = β1 * T1
δT2 = δ2 * T2
βT2 = β2 * T2
two_cylinders_δT = δT1 + (δ2 - δ1) * T2
two_cylinders_βT = βT1 + (β2 - β1) * T2

Prop1 = xri.sim.propAS(δT1, βT1, E1, z_final, delta_x, supersample=3)
Prop2 = xri.sim.propAS(δT2, βT2, E1, z_final, delta_x, supersample=3)
Prop3 = xri.sim.propAS(two_cylinders_δT, two_cylinders_βT, E1, z_final, delta_x, supersample=3)
```

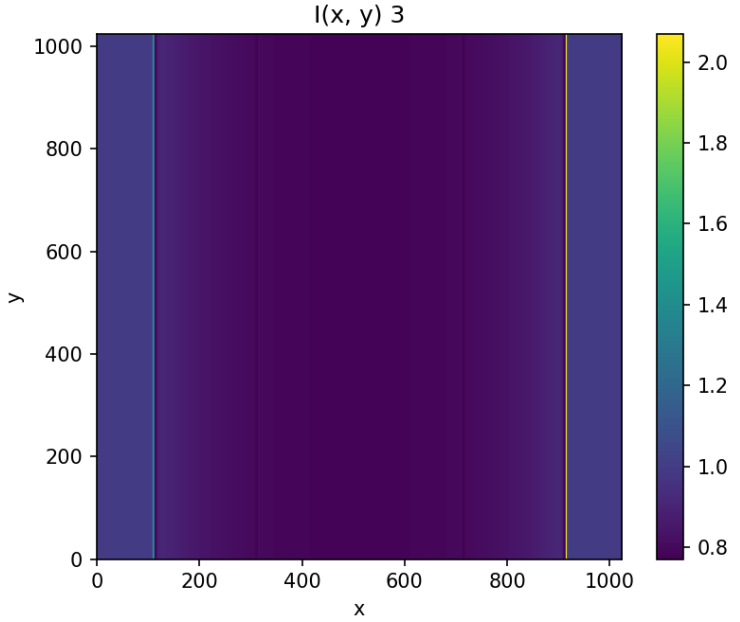


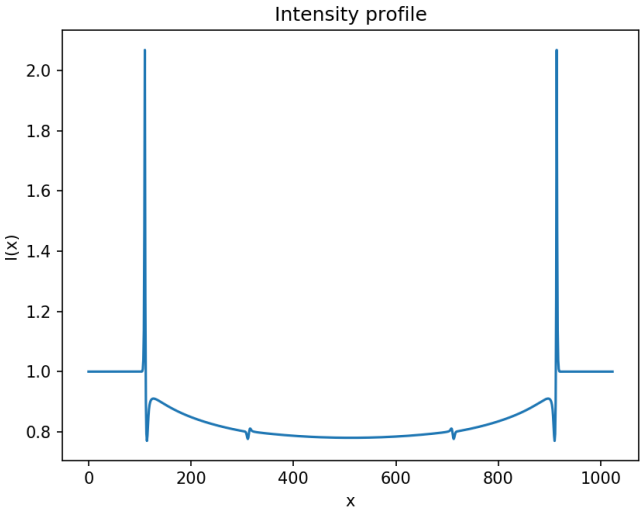
Prop2 = xri.sim.propAS(δT2, βT2, E1, z_final, delta_x, supersample=3)





Prop3 = xri.sim.propAS(two_cylinders_δT, two_cylinders_βT, E1, z_final, delta_x, supersample=3)





Success!

The composition of the brain

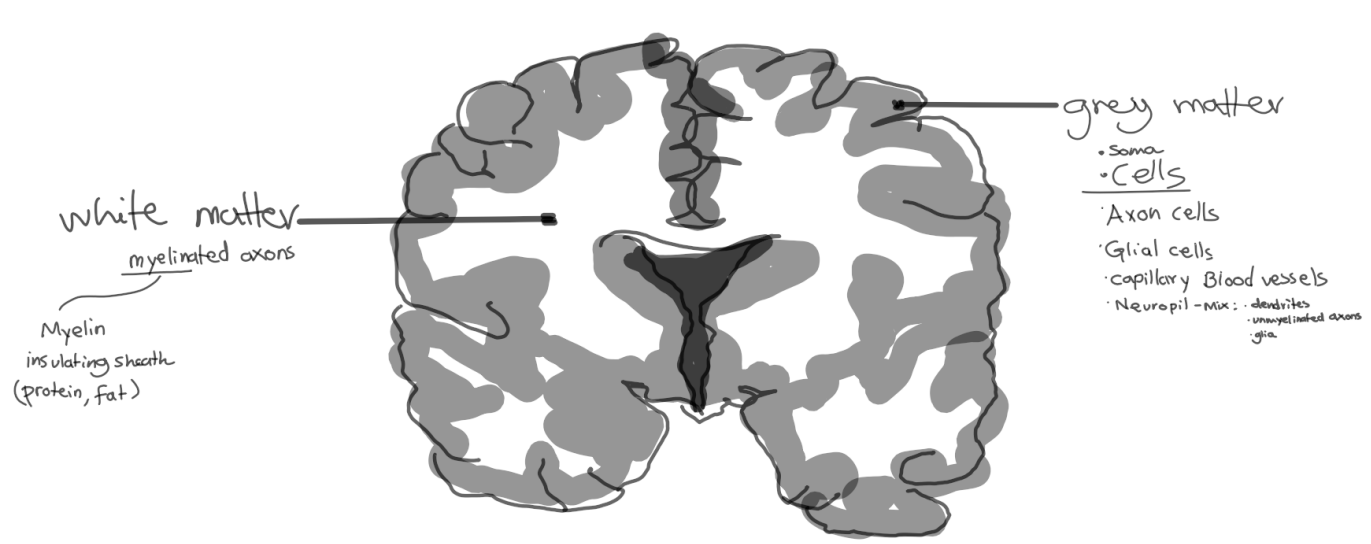
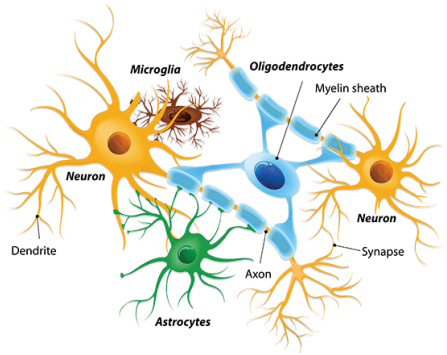


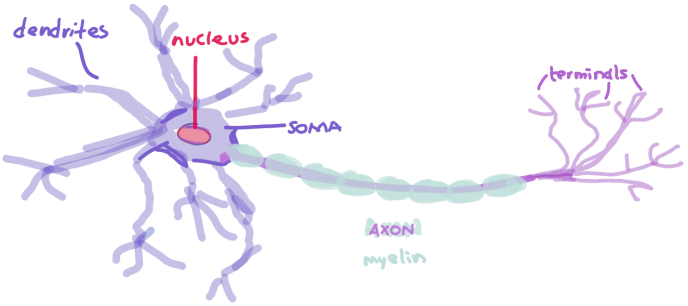
Diagram based on (Mackenzie)

glia
Non-nerve cells that make up about half of the cells in the brain. Some glial cells wrap around axons. This speeds the rate of neural signaling and helps prevent confusing “cross-talk” between neighboring nerve cells. Other glial cells provide nutrients and oxygen to neurons, or defend the nervous system from injury.



(Brookshire)

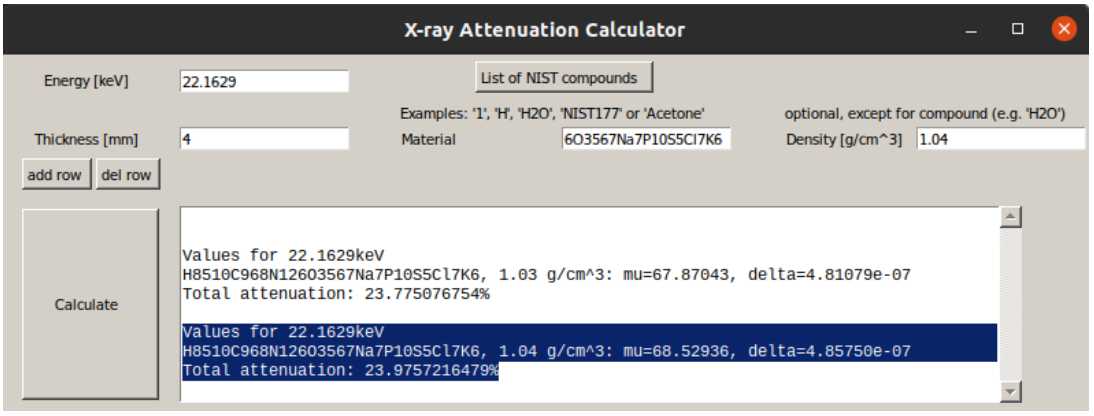
Neuron
An impulse-conducting cell. Such cells are found in the brain, spinal column and nervous system. (Brookshire)



I asked Linda for some advice and she explained how to use the **Tissues** spreadsheet made by MK

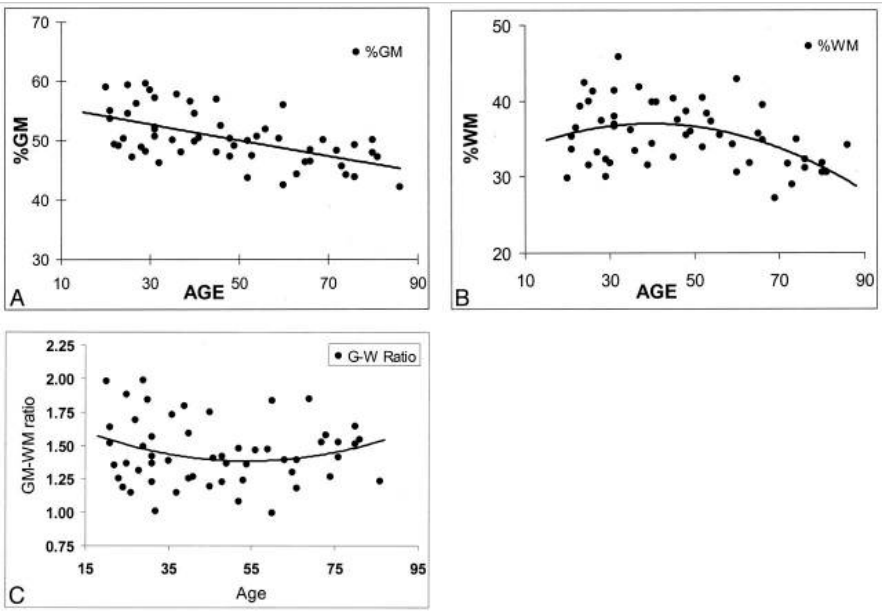
A	B	C	D	E	F	G	H	I	J	K	L	M	
Compound	Orange X-ray Book: Above 30eV atomic scattering factors can explain optical properties (1-44).												
Tissue Equiv	Density (g/cm^3)	Element	Weight Frac	RAM	Mol	Mol Frac		Caution: tissue values	may be good for absorption only				
BRAIN, GREY/WHITE MATTER (ICRU-44)													
ICRU-44	**1.04**	H	0.107	1.00794	0.10615711253	0.64438567075	0.0003786012	1702.017	3404.034	5106.051	6808.0679	8510.0849	H
		C	0.145	12.011	0.01207226709	0.07328002562		193.55466	387.10932	580.66397	774.21863	967.77329	C
		N	0.022	14.00674	0.0015706724	0.00953415901		25.182591	50.365181	75.547772	100.73036	125.91295	N
		O	0.712	15.9994	0.04450166881	0.2701301592		713.49526	1426.9905	2140.4858	2853.981	3567.4763	O
		Na	0.002	22.989768	8.6995223E-05	0.00052807084		1.3947944	2.7895888	4.1843832	5.5791777	6.9739721	Na
		P	0.004	30.973762	0.00012914156	0.00078390388		2.0705267	4.1410533	6.21158	8.2821066	10.352633	P
		S	0.002	32.066	6.2371359E-05	0.0003786012		1	2	3	4	5	S
		Cl	0.003	35.4527	8.4619789E-05	0.00051365168		1.3567091	2.7134182	4.0701272	5.4268363	6.7835454	Cl
		K	0.003	39.0983	7.6729679E-05	0.00046575782		1.2302069	2.4604139	3.6906208	4.9208278	6.1510347	K

I can use column M to create the chemical formula
$H_{8510}C_{968}N_{126}O_{3567}Na_7P_{10}S_5Cl_7K_6$



Back-of-the-envelope calculation

What are the proportions of gray and white matter in a normal adult (<50 years old) brain??

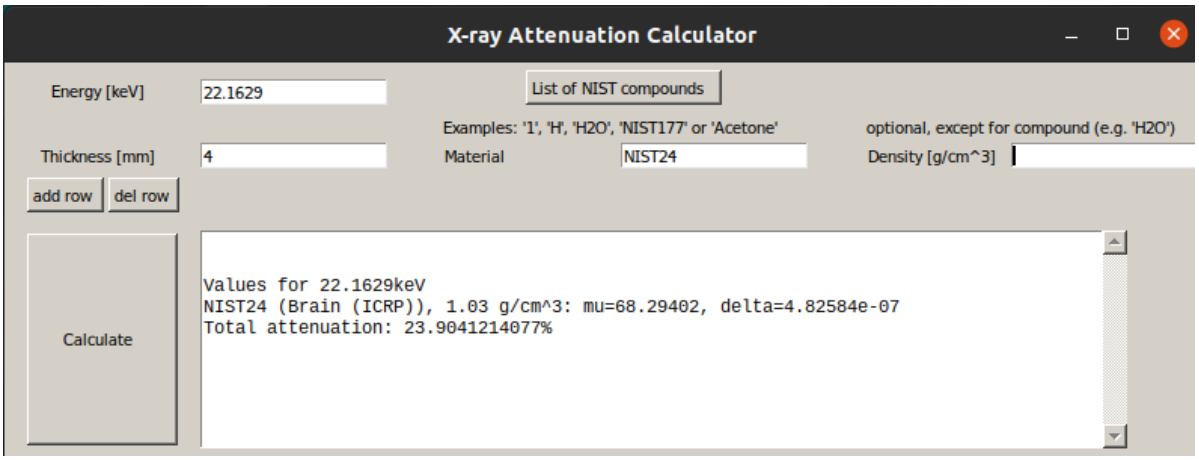


Regression analysis of fractional brain tissue volume estimates on age in 54 healthy adult subjects. Linear and weighted constrained quadratic models are presented; these indicate the age-related volume estimates throughout adulthood in normal brains. *A*, %GM. *B*, %WM. *C*, GM/WM ratio (Yulin et al. Fig 2.).

Let's say that we are looking at a **~30 year old**.
Then we have %GM/%WM ~ 52% / 35% = 1.49
The average brain weight of the adult male was **1336** gr; for the adult female **1198** gr. With increasing age, brain weight decreases by 2.7 gr in males, and by 2.2 gr in females per year (Hartmann et al. Abstract)
I am going to assume the same ratio of GM/WM in both sexes because the internet is full of bias with respect to this and I am **angry** that this is even an issue.

Male brain weight : 1336 g GM = **694.72** g WM = **467.6** g	**Female brain weight : 1198** g GM = **622.96** g WM = **419.3** g

Brain data already in the X-ray attenuation calculator



Then $\delta_{brain} = 0.52\,\delta_{GM} + 0.35\,\delta_{WM} + 0.13\,\delta_{other}$
And similarly for μ_{brain}
Which leaves me with: a single equation with 3 unknowns which I am unsure about how to solve.
Maybe if I find the refractive index n of each material?

12/09/21
TO THE INTERNET!
I found this paper: **High-resolution tomographic imaging of a human cerebellum: comparison of absorption and grating-based phase contrast**

The Gaussian fit (figure 2b) was also used to quantify the values of the decrement of the real part of the refractive index for the features inside the human cerebellum plus formalin. Here the values $\delta_{form} = (0.50 \pm 0.03) \times 10^{-8}$ for the formalin, $\delta_{WM} = (1.61 \pm 0.08) \times 10^{-8}$ for the white matter (Schulz et al. 1670)

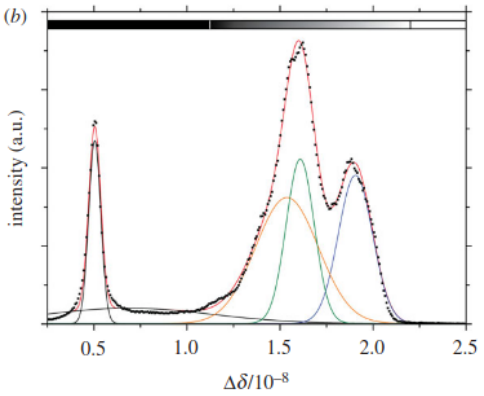


Figure 2. One reconstructed slice plus the appropriate histogram of the phase-contrast results showing three different strata and several blood vessels (bright regions in the slices that exceed the grey-scale range) of the human cerebellum. The mean value of the blood vessels amounts $\Delta\delta = 3.6 \times 10^{-8}$. The formalin peak (left peak in the diagram) has the value $\Delta\delta = 5.03 \times 10^{-9}$ with the standard deviation $\sigma_{\text{form}} = 0.32 \times 10^{-9}$. Thus, the grey-scale range corresponds to 34 standard deviations of the formalin peak. The standard deviation is a measure of the homogeneity of the tissue or solution: the narrower the width the Gaussian is, the more homogeneous the substance. (*a*) Scale bar, 1 mm. (*b*) Orange line, stratum moleculare (a); green line, white matter (b); navy blue line, stratum granulosum (c).

And also this paper: **Interface-specific x-ray phase retrieval tomography of complex biological organs**
According to the NIST database (http://www.nist.gov/index.html), grey and white matter (the main tissue types present in the brain) are virtually identical in reference to diagnostic energy x-ray interactions and are herein treated as identical.

SHOULD I TREAT δ_{WM} and δ_{GM} as identical for now?

all materials in the brain sample effectively refract and attenuate x-rays to a similar degree; hence, the sample behaves somewhat like a single-material object and thuss we need to utilize equation (1) only (see figure 2). We used the δ and μ values for grey/white matter listed in table 2. The same slice in figure 8(b) is shown in figure 9, now with the phase retrieval process included. Despite the subtle differences in complex refractive index, the tomogram yields clearly demarcated tissue borders at the grey/white matter boundaries(Beltran et al. 7365).

I highlighted the last sentence because I am curious about this. Why are there obvious boundaries visible between the materials given that they are assumed to have similar optical properties?

Linear attenuation coefficients of grey and white matter

Looking around the internet once again I found this paper:
"Attenuation Coefficients of Various Body Tissues, Fluids, and Lesions at Photon Energies of 18 to 136 keV" https://sci-hub.se/10.1148/117.3.573
It comes with a big table of linear attenuation coefficients of grey and white matter in human brains at different energy values.

Table IV: Linear Attenuation Coefficients as a Function of Photon Energy for Human and Monkey Gray Matter, White Matter, and Mixed Brain Tissue

	Linear Attenuation Coefficients (cm^{-1})									
	Monkey						Human			
Energy (keV)	Gray matter (10)* Exp.	Calc.	Mixed Brain† (3) Exp.	Calc.	White matter (10) Exp.	Calc.	Gray matter (4) Exp.	Calc.	White matter (4) Exp.	Calc.
17.7	1.124	1.124	1.101	1.101	1.091	1.090	1.122	1.122	1.092	1.092
21.1	0.7351	0.7338	0.7241	0.7249	0.7111	0.7163	0.7310	0.7333	0.7106	0.7119
26.4	0.4722	0.4721	0.4664	0.4660	0.4610	0.4612	0.4710	0.4726	0.4593	0.4602
27.4	0.4435	0.4431	0.4378	0.4378	0.4347	0.4335	0.4445	0.4442	0.4340	0.4330
31.1	0.3668	0.3658	0.3626	0.3622	0.3591	0.3590	0.3683	0.3675	0.3577	0.3593
35.5	0.3069	0.3091	0.3049	0.3076	0.3047	0.3054	0.3125	0.3114	0.3062	0.3057
41.4	0.2653	0.2649	0.2647	0.2646	0.2632	0.2633	0.2667	0.2667	0.2631	0.2633
47.2	0.2380	0.2395	0.2388	0.2392	0.2379	0.2384	0.2386	0.2405	0.2387	0.2384
52.0	0.2251	0.2255	0.2252	0.2250	0.2250	0.2244	0.2264	0.2260	0.2251	0.2245
59.5	0.2107	0.2104	0.2079	0.2096	0.2078	0.2088	0.2096	0.2104	0.2080	0.2090
59.6	0.2115	0.2102	0.2103	0.2095	0.2086	0.2087	0.2117	0.2103	0.2079	0.2089
84.3	0.1832	0.1830	0.1836	0.1832	0.1826	0.1823	0.1843	0.1838	0.1821	0.1820
97.4	0.1739	0.1741	0.1750	0.1748	0.1749	0.1742	0.1747	0.1753	0.1741	0.1738
103.2	0.1702	0.1709	0.1709	0.1716	0.1705	0.1712	0.1709	0.1716	0.1711	0.1709
121.9	0.1626	0.1624	0.1627	0.1626	0.1622	0.1626	0.1633	0.1630	0.1630	0.1632
136.3	0.1576	0.1573	0.1565	0.1564	0.1578	0.1571	0.1578	0.1570	0.1586	0.1583

* Number of samples measured.
† Approximately 45% white matter and 55% gray matter.

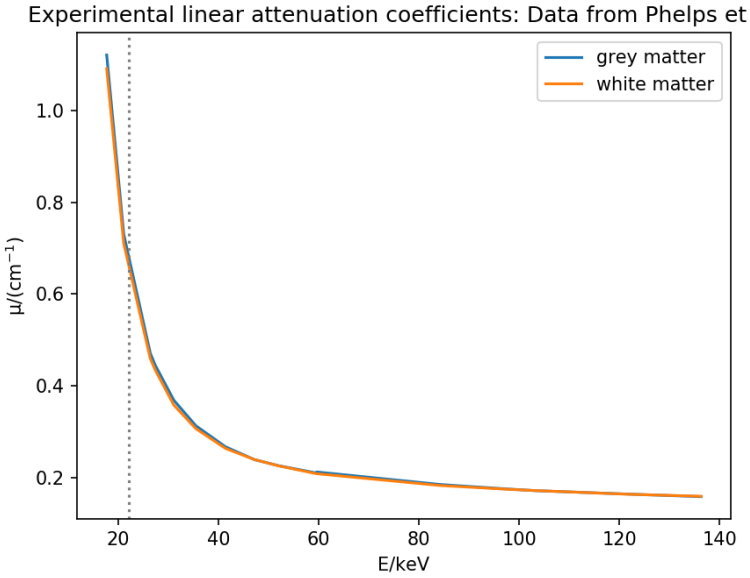
(Phelps et al. 557)

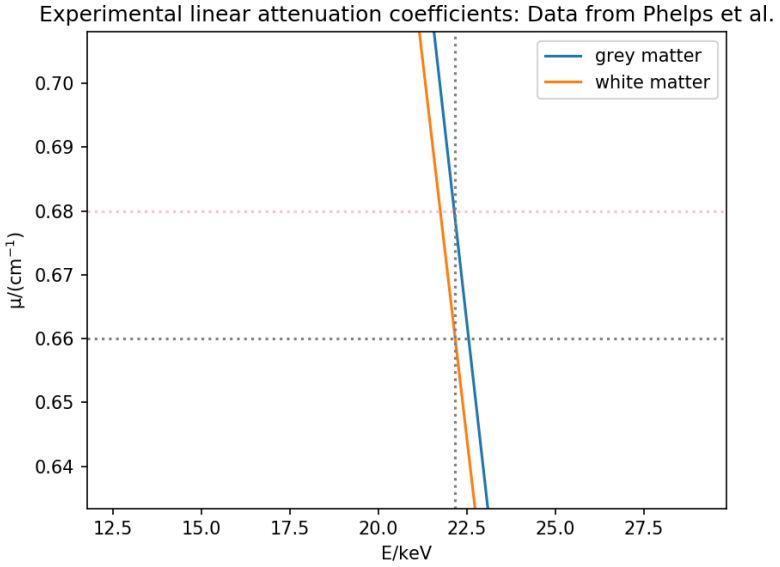
Unfortunately the energy values don't match mine. So I need to interpolate this data. I am exclusively going to use the human experimental data in this table.

```python
import csv
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 150

f = open('experimental_mu_brain.csv')
csv_f = csv.reader(f)

Energy = []
GM = []
WM = []

for row in csv_f:
    # print(row)
    E = float(row[0]) # keV
    Energy.append(E)
    mu_gm = float(row[1])
    GM.append(mu_gm)
    mu_wm = float(row[2])
    WM.append(mu_wm)

# print(Energy)

plt.plot(Energy, GM, label='grey matter')
plt.plot(Energy, WM, label='white matter')
plt.axvline(x=22.1629, color='gray', linestyle=':')
plt.xlabel("E/keV")
plt.ylabel(r"$\mathrm{\mu/(cm^{-1})}$")
plt.title("Experimental linear attenuation coefficients: Data from Phelps
plt.legend()
plt.show()
```

Looks like

$\mu_{GM} = 0.68\ cm^{-1}$ and $\mu_{WM} = 0.66\ cm^{-1}$ pretty much.





Week 8 (13/09/21 –19/09/21)

Aims:

1. ~~Create simulations of phase contrast imaging through different densities and materials~~
2. ~~Verify if simulations yield observable fringes in lab conditions~~

Tasks:

1. Meetings on **Monday 13/09** and **Wednesday 15/09**
    a. ~~Group: **1 pm (got confused)**~~
    b. ~~one-on-one: **11 am**~~
2. **Read PyQt GUI book**
3. **Plan progress report wk 9**

13/09/2021

MK's latest comments

*...grey and white matter are extremely similar in their refractive index properties. However, there is a small difference in mu between the two - otherwise we wouldn't be able to see much of any contrast in a brain CT. So we know they are small, which is why we said that in the Beltran paper and because we haven't really observed fringes. **I always assumed this was because it's really just a density difference which does create absorption contrast in CT, but should not create phase contrast if density really does cancel out.** I think your water/ice simulation proves there probably really are fringes, just very small ones (...I'm pretty sure these fringes would be too small to see when you add in a point-spread function and noise. What distance was this? You could simulate that to see when they would disappear - but we can talk about that later)*

*Linda's data can actually tell you exactly what you need. **This is the point of your project***
*I want to know if there actually are small fringes there that we have been ignoring because they were underwhelming. From her CT data she could measure the mu values in regions of grey/white matter. **You can then assume they are the same material with just a density difference. You can then use this to figure out how much the density changed,** hence how much to change delta by the same factor.*

Linda's Data

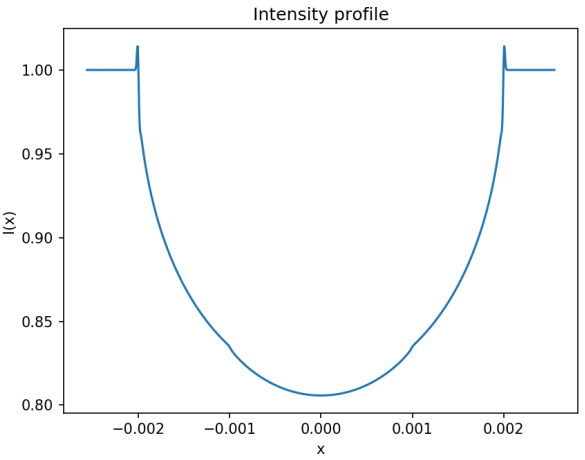
These values can vary quite a bit from experiment to experiment. I've just looked at some SPring-8 data from 2017B. **The grey matter values are around 52, while the white matter values are more like 55-57**.
The data was acquired at **24keV** and a sample-to-detector distance of **5m**. The source-to-sample distance at SPring-8 BL20B2 is 210m. It was reconstructed from 1800 projections, and a ring correction was applied.

Small test for fun

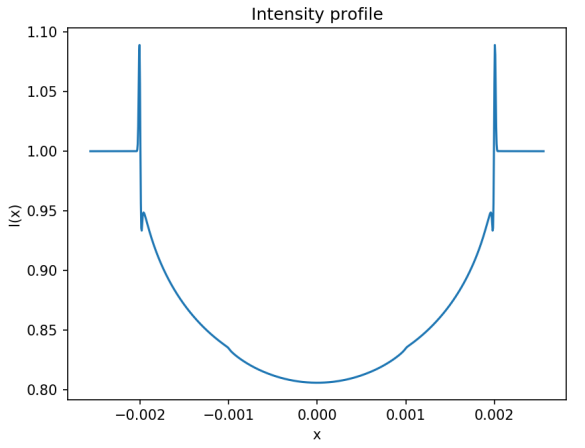
```
    # Parameters Linda's data
    E1 = 24 # keV
    λ = h * c / (E1 * 1000 * const.eV)
    k1 = 2 * np.pi / λ  # x-rays wavenumber

    # # Material = gray matter, density = ? g/cm**3
    δ1 = 16.1 * nm # ?
    μ1 =  52 # per m
    β1 = μ1 / (2 * k1)
    # # Material = white matter, density = ? g/cm**3
    δ2 = 16.1 * nm # ?
    μ2 = 56 # per m
    β2 = μ2 / (2 * k1)
```

Here I am using $\delta = 16.1$ mm as my **first approximation to the RID**

z_final = 1 * m



z_final = 5 * m



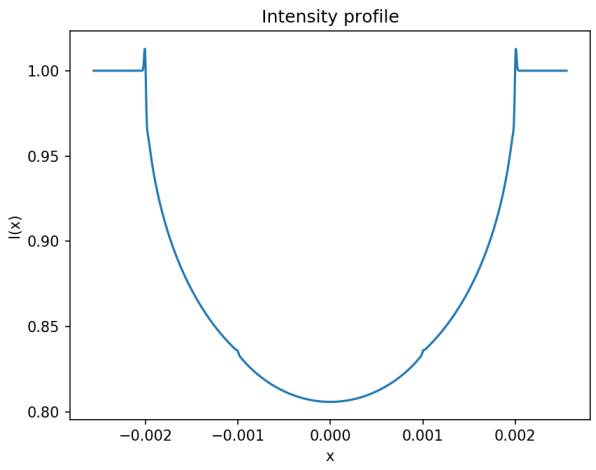
Sensei MK says:
You can then assume they are the same material with just a density difference. You can then use this to figure out how much the density changed, *hence how much to change delta by the same factor.*

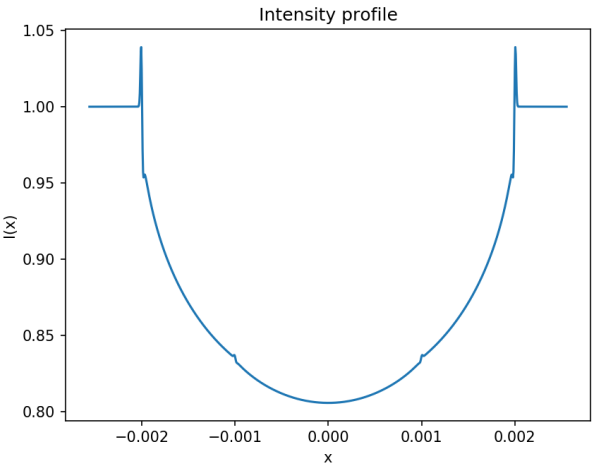
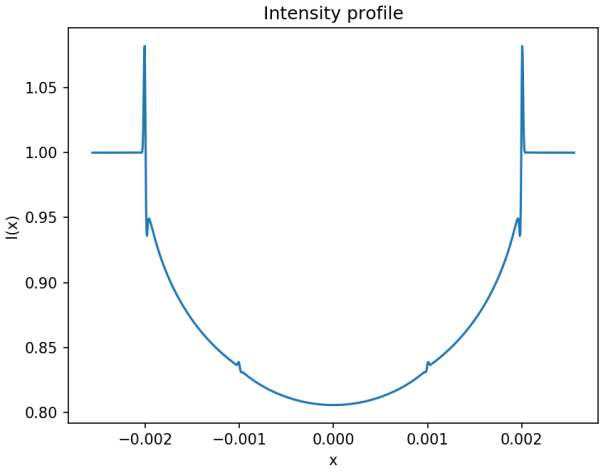
So if the materials are the "same" , $\mu_{GM} = 52$, $\mu_{WM} = 56$ and I found that $\delta_{WM} = (1.61 \pm 0.08) \times 10^{-8}$ using (Schulz et al. 1670)

Then $\delta_{GM} = \frac{52}{56}(1.61 \times 10^{-8}) = 1.49 \times 10^{-8}$

```
    # # Material = gray matter, density = ? g/cm**3
    δ1 = (52 / 56) * 16.1 * nm # ?
    μ1 =  52 # per m
    β1 = μ1 / (2 * k1)
    # # Material = white matter, density = ? g/cm**3
    δ2 = 16.1 * nm # ?
    μ2 = 56 # per m
    β2 = μ2 / (2 * k1)
```

z_final = 1 * m



z_final = 2.5 * m	
z_final = 5 * m	

14/09/2021

MK's latest comments

I don't think the delta values from Schulz can be right. Using Florian's calculator for brain tissue (NIST24) at 24 keV, the value is around 4.17E-7. This is significantly larger than your Schulz's values, so the phase contrast should be a lot stronger, which is good.
Interestingly, I googled brain density and found this website:https://itis.swiss/virtual-population/tissue-properties/database/density/

Density

The following table contains values for the density of all tissues, including statistical information on the standard deviation and the spread in the values.

Note that if two values are drawn from the same publication, there will be a difference between the number of studies indicated in the table below and the number of references provided in the downloadable reference table.

Temperature-dependent density values for air are available in a separate table at the bottom of this page.

Density (kg/m³)	Average	Standard Deviation	Number of Studies	Minimum	Maximum
Brain	1046	6	2	1041	1050
Brain (Grey Matter)	1045	8	2	1039	1050
Brain (White Matter)	1041	2	3	1040	1043

(Hasgall et al.)

It shows people have measured differences between grey and white matter, which should prove useful. You could actually just use these densities for the delta calculations. I hope the ratio is the same as the attenuation ratio that Linda sent through, but I didn't calculate it. Please check to see.

Here I am verifying that Linda's data matches the densities from the website you sent me, I am doing a ratio check in each of these table cells.
The first row of the table is a check of mu (from Florian's X-ray att calc. NIST24 value) divided by density (the density value corresponds to the values in the itis website).
The second row is a check of mu (from Florian's X-ray att calc. NIST24 value) divided by Linda's experimental mu values, in the case of the deltas I simply multiplied Florian's X-ray att calc. NIST24 value by the corresponding (grey or white matter) mu ratios.

Calculating δ_G, δ_W & verifying Linda's μ_G μ_W with itis densities

E = 24 keV

μ_B = 58.0189 $\quad \rho_B = 1.04\ g cm^{-3}$

μ_G = 52 (Linda's data) $\quad \rho_G = 1.045 g cm^{-3}$ (itis data)

μ_W = 56 (Linda's data) $\quad \rho_W = 1.041 g cm^{-3}$ (itis data)

$\delta_B = 4.11479 \times 10^{-7}$

δ_G = ?

δ_W = ?

	μ_G	μ_W	$\delta_G (\times 10^{-7})$	$\delta_W (\times 10^{-7})$
$\frac{\mu}{\rho}$	58.2978 *	58.0747 *	4.1345	4.1187
$\frac{\mu_B}{\mu}$	52 *	56 *	4.591	4.2631

* Linda's values do not suit itis densities

The possible values suit 2 cases.

1. The pessimistic case: where I use the attenuation calculator data and the densities from itis showing values much more similar to each other.
2. The optimistic case where I use Linda's and the attenuation calculator data: **this case shows much more difference in all the values.**

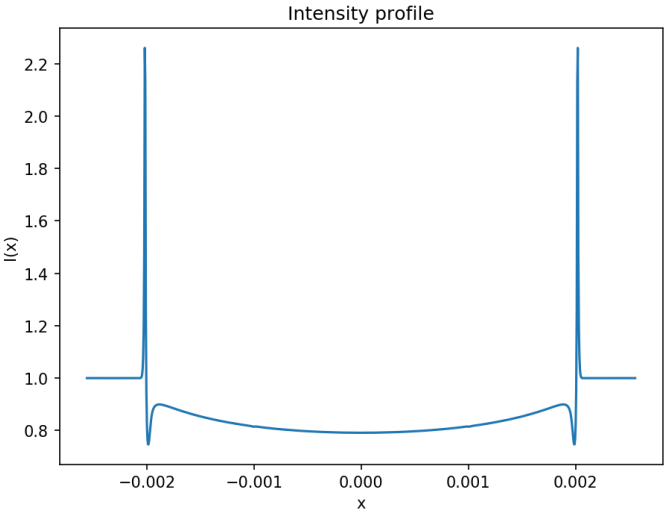
CASE 1

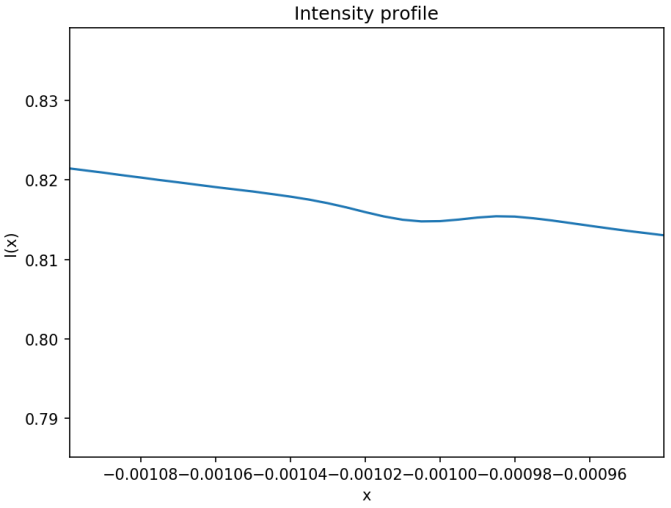
```
# Pessimistic case
E1 = 24 # keV
λ = h * c / (E1 * 1000 * const.eV)
k1 = 2 * np.pi / λ  # x-rays wavenumber

# # Material = gray matter, density = 1.045 g/cm**3
δ1 = 413.45 * nm
μ1 =  58.2978 # per m
β1 = μ1 / (2 * k1)
# # Material = white matter, density = 1.041 g/cm**3
δ2 = 411.87 * nm
μ2 = 58.0747 # per m
β2 = μ2 / (2 * k1)
```

T = gaussian_filter(T, sigma=3)

z_final = 2.5 * m

Extremely small fringes, but fringes nonetheless!



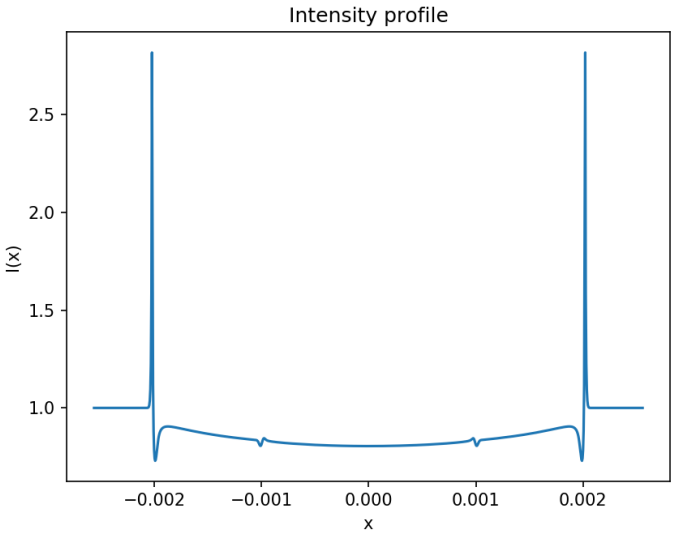


CASE 2

```
# Optimistic case
E1 = 24 # keV
λ = h * c / (E1 * 1000 * const.eV)
k1 = 2 * np.pi / λ  # x-rays wavenumber

# # Material = gray matter, density = ? g/cm**3
δ1 = 459.1 * nm
μ1 =  52 # per m
β1 = μ1 / (2 * k1)
# # Material = white matter, density = ? g/cm**3
δ2 = 426.31 * nm
μ2 = 56 # per m
β2 = μ2 / (2 * k1)
```

T = gaussian_filter(T, sigma=3)

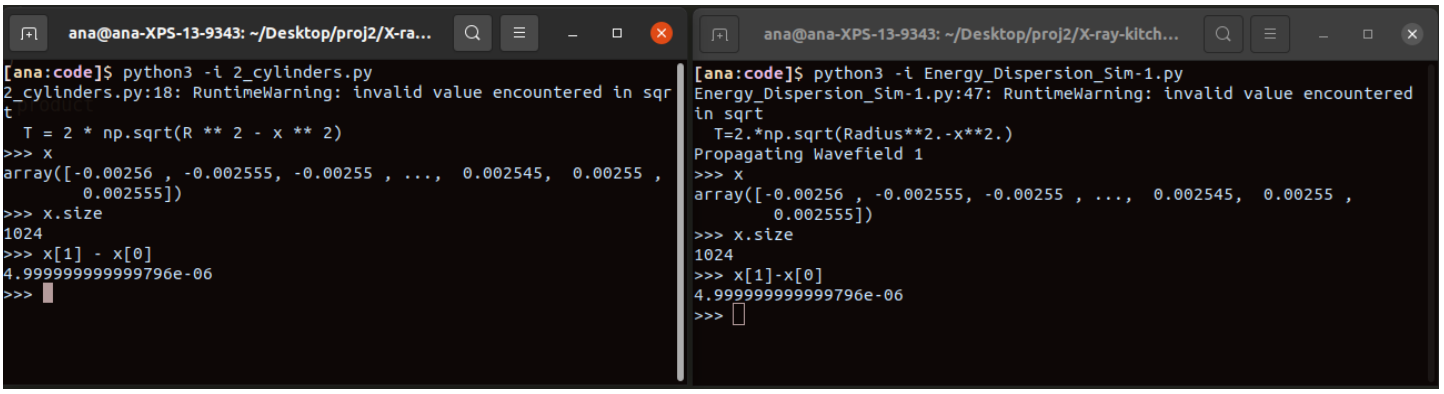
z_final = 2.5 * m



Clear fringes visible!	

...the next step you should do is rebin (resample) the data to a pixel size that we'd normally use. At the highest resolution we use pixels of 6.5um. At the lowest it's ~20um. Then the resolution is limited by the PSF, so you should blur the data with a Gaussian with a standard deviation of around 1.3 pixels wide (giving a FWHM of ~3 pixels). ***This will help us to see if the little peaks can actually be resolved in a real experiment.***

I am pretty certain that the resolution matches an adequate size to be resolved in the lab.
Here is a comparison of the x array of both MK's Energy_Dispersion_sim-1.py and my code. We both use a pixel size of 5um



15/09/21

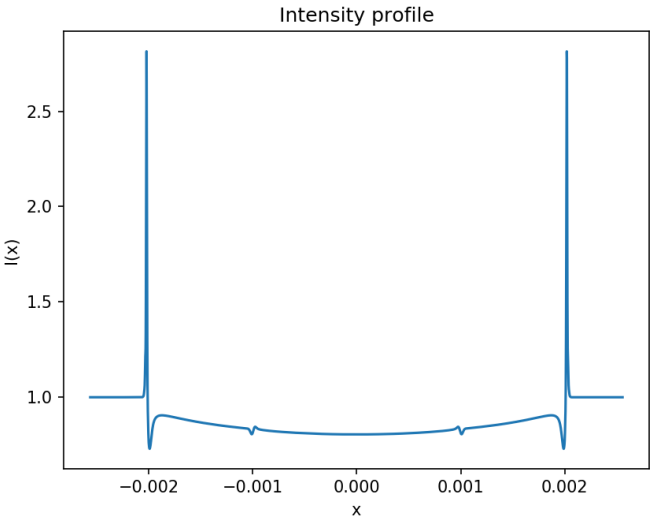
MEETING DAY!

MK's latest comments

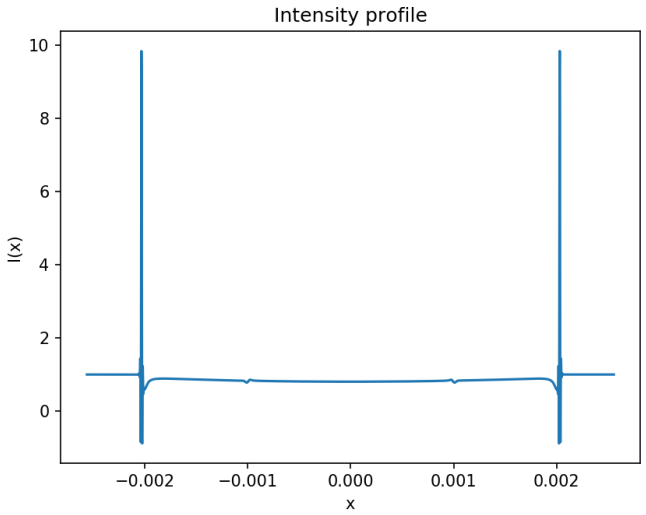
> *It's interesting to see that the theoretical and experimental values don't match so well, but that may also be because we are assuming there is no chemical difference in the theory version, only a density difference. This assumption is unlikely true. Of course theory never matches experiment perfectly!*
> *I think that we are better off using Linda's experimental values for mu and the equivalent delta values. I think this is better to use* ***because it's real experimental data for the samples we image and it should help compensate for chemical composition differences too.*** *In that case it's good to see that there could actually be a faint set of fringes between the materials with a pixel size of 5um. If you can compute the blurred version of this and do so at distances up to 5m, that would be great.*

OPTIMISTIC CASE &
# # x-array parameters
n = 1024
n_x = n
# # Blurry version
x_max = (n_x / 2) * 5 * um
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
delta_x = x[1] - x[0]
# # y-array parameters
n_y = n
# # Blurry version
y_max = (n_y / 2) * 5 * um
y = np.linspace(-y_max, y_max, n_y, endpoint=False)
delta_y = y[1] - y[0]
y = y.reshape(n_y, 1)

#Re-bin data with scipy.ndimage.zoom each pixel is now 20um
I = zoom(I, 4.0, order=3)
x = zoom(x, 4.0, order=3)
z_final = 2.5 * m



z_final = 5 * m



Looks pretty wiggly

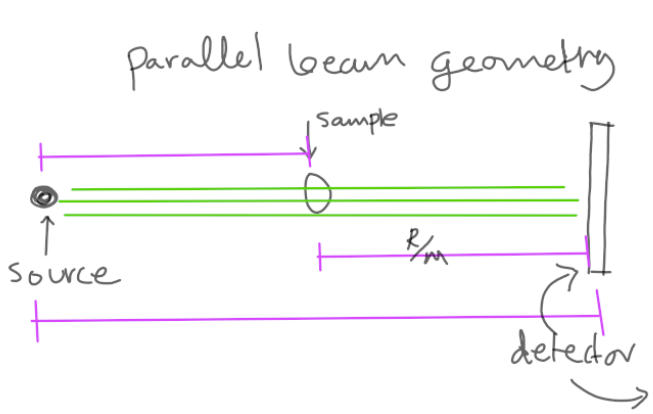
*Regarding lab experiments, our best bet is to try the ice/water combo with the greatest resolution/most phase contrast and best SNR we can get. In that case we should use the photon counting detector, which has **55um** pixels. We should probably use a **magnification between 2.5 and 4.0 to get decent resolution**. This detector has a PSF of only 1 pixel wide, so that will help us see any tiny fringes. You may be aware then that the problem of magnification is that the effective propagation distance is the actual propagation distance divided by magnification. The maximum source-to-detector distance in my lab is ~2.5m. For the different magnifications I've suggested, **you could calculate the effective propagation distance and pixel size** and run your simulations with those parameters. You don't need to actually magnify the images, this rescaling pulls it back to being equivalent to a parallel beam geometry.*

Effective propagation distance & pixel size calculations

1. $P_{eff} = \frac{P}{M} = \frac{2.5\,m}{2.5\,X} = 1\,m$

2. $P_{eff} = \frac{2.5\,m}{4.0\,X} = 0.625\,m$

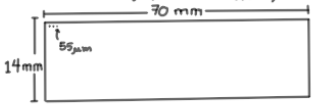
Having to match the apparatus parameters means that I do not have the opportunity of choosing freely the number of points in my x and y arrays. Since the photon counting detector we will be using, has **55um** sized pixels then



	detector plane	object plane
Δx	$55_{\mu m}$	$\frac{55_{\mu m}}{M}$
Δy	$55_{\mu m}$	$\frac{55_{\mu m}}{M}$

M - magnification

The number of pixels (n_x, n_y) depends on photon detector dimensions



$n_x = int((x_{max} - x_{min})/\Delta x)$

$n_y = int((y_{max} - y_{min})/\Delta y)$

where $x_{max} = \frac{70\,mm}{2} = 35\,mm$

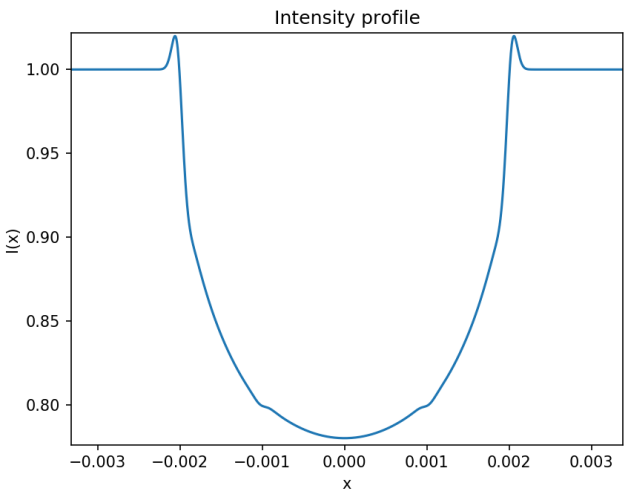
$x_{min} = -x_{max}$

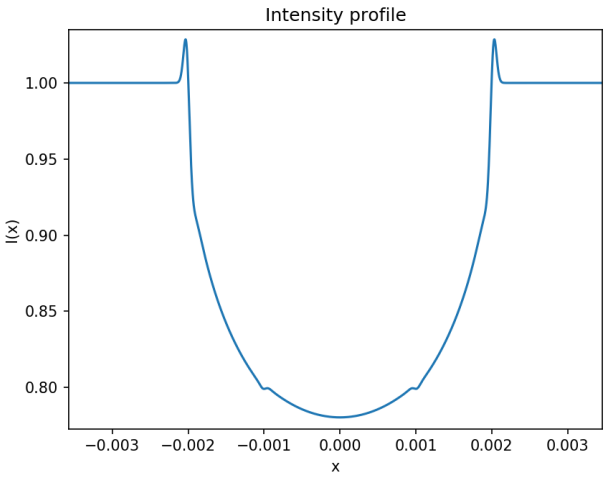
$y_{max} = \frac{14}{2}\,mm = 7\,mm$

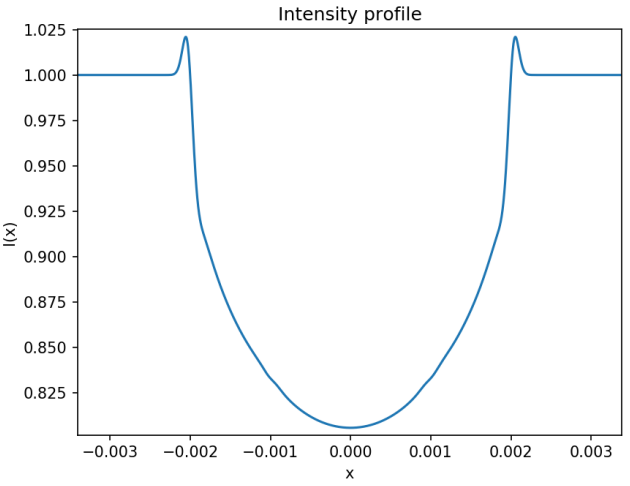
$y_{min} = -y_{max}$

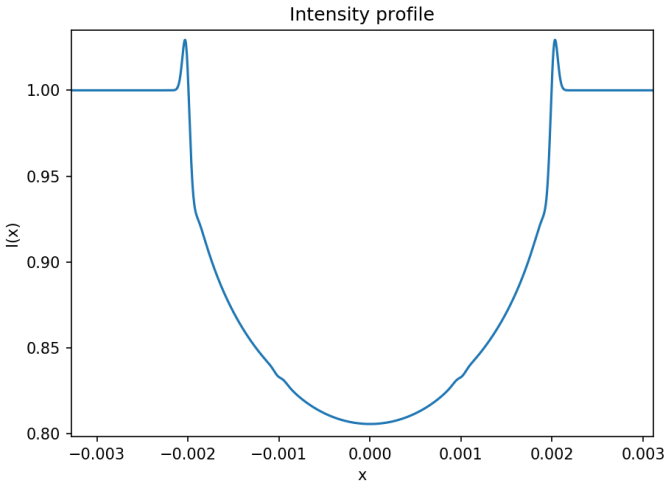
#Testing water / ice

```
# Magnification
M = 2.5
# # x-array parameters
delta_x = 55 * um / M
x_max = 35 * mm / M
x_min = -x_max
n_x = int((x_max - x_min) / delta_x)
print(f"\n{n_x = }")
x = np.linspace(-x_max, x_max, n_x, endpoint=False)

 # y-array parameters
delta_y = 55 * um / M
y_max = 7 * mm / M
y_min = -y_max
n_y = int((y_max - y_min) / delta_y)
print(f"\n{n_y = }")
y = np.linspace(-y_max, y_max, n_y, endpoint=False).reshape(n_y, 1)

z_final =  2.5 * m / M # eff propagation distance
```

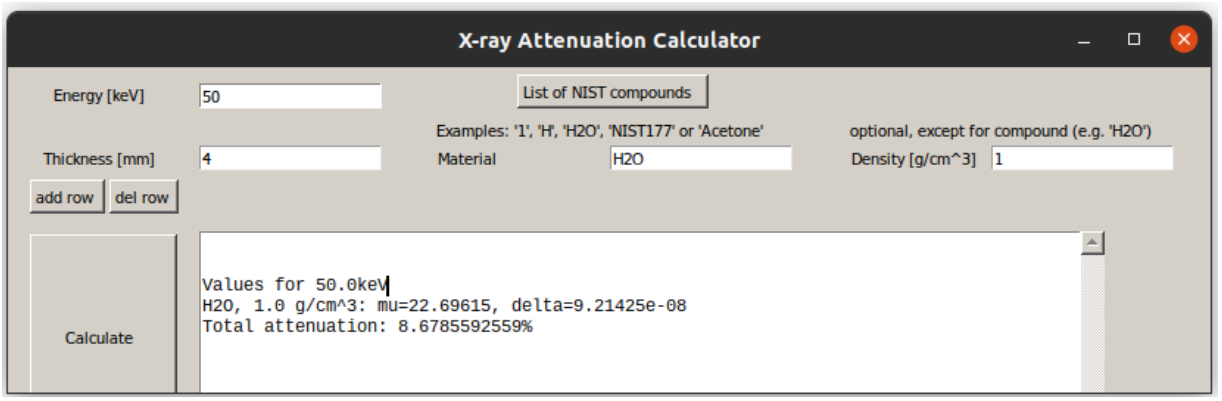


M = 4.0	

Testing # grey / white matter - Optimistic case
 	
# Magnification 	
M = 2.5
# # x-array parameters 	
delta_x = 55 * um / M
x_max = 35 * mm / M
x_min = -x_max
n_x = int((x_max - x_min) / delta_x)
print(f"\n{n_x = }")
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
 	
# y-array parameters 	
delta_y = 55 * um / M
y_max = 7 * mm / M
y_min = -y_max
n_y = int((y_max - y_min) / delta_y)
print(f"\n{n_y = }")
y = np.linspace(-y_max, y_max, n_y, endpoint=False).reshape(n_y, 1)
 	
z_final = 2.5 * m / M # eff propagation distance
	

M = 4.0	

Testing TIE - RK method at higher energy



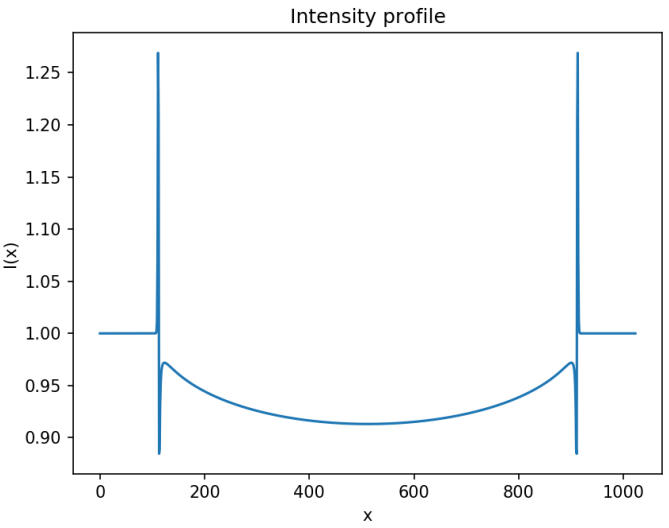
MY METHOD (second_proj_approx_2D.py)	

Note: cylinder is not centered at zero.

```
# sample: H20 density: 1.0 g/(cm**3)
energy1 = 50 * keV
δ1 = 92.1425 * nm
μ1 = 22.69615
λ = h * c / energy1
k0 = 2 * np.pi / λ  # x-rays wavenumber
```

 # Blurring
σ_x = 0.0027 * mm ←Maybe this blurring is to small to be correct?
Number of pixels used in blurring: $\sigma \times (n_x/(x_{max} - x_{min}))$ = 0.138

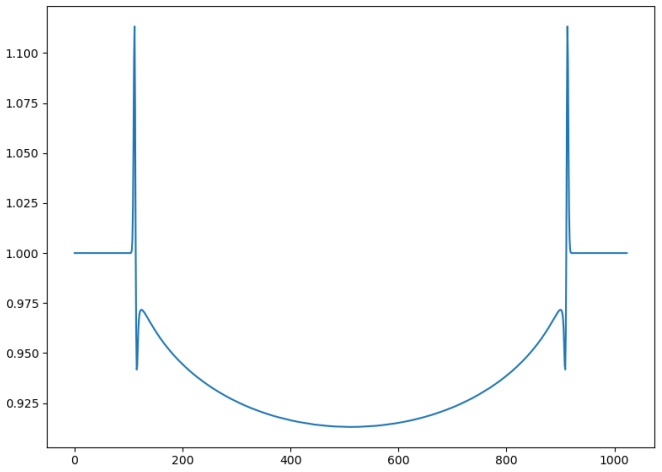
```
D = 4 * mm
R = D / 2
height = 10 * mm

h = const.h # 6.62607004e-34 * J * s
c = const.c # 299792458 * m / s

# array parameters & z evolution
n = 1024
x_max = (n_x / 2) * 5 * um
x = np.linspace(-x_max, x_max, n, endpoint=False)
delta_x = x[1] - x[0]
n_y = n
y_max = (n_y / 2) * 5 * um
y = np.linspace(-y_max, y_max, n, endpoint=False)
delta_y = y[1] - y[0]
y = y.reshape(n_y, 1)
z_final = 1 * m
delta_z = 1 * mm  # (n_z = 1000)
```



Plot appears very smooth and peaks look decent

MK/Florian's Method

```
# sample: H20 density: 1.0 g/(cm**3)
energy1 = 50
delta1 = 92.1425e-9
k1=xri.utils._conversions.energy2k(energy1)
mu1 = 22.69615
beta1=mu1/(2.*k1)

xsize=1024
ysize=xsize
pxl=5E-6
z=1. # Prop distance, m

x=np.arange(-xsize/2,xsize/2)*pxl
Radius=2.E-3
ones_y=np.ones(ysize)
#Expand 1D to 2D with outer product
T=np.outer(ones_y,T)
```



Another aspect that is different between our code is the way I calculate the ICs with a brute force integral over the cylinder while MK only uses **T(x, y)**

```
def phase(x, y):
        # phase gain as a function of the cylinder's refractive index
        z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)
        dz = z[1] - z[0]
        # Euler's method
        Φ = np.zeros_like(x * y)
        for z_value in z:
        print(z_value)
        Φ += -k0 * δ(x, y, z_value, δ1) * dz
        return Φ # np.shape(Φ) = (n_y, n_x)


def BLL(x, y):
        # TIE IC of the intensity (z = z_0) as a function of the cylinder's attenuation coefficient
        z = np.linspace(-2 * R, 2 * R, 2 ** 12, endpoint=False)
        dz = z[1] - z[0]
        # Euler's method
        F = np.zeros_like(x * y)
        for z_value in z:
        print(z_value)
        F += μ(x, y, z_value, μ1)* dz
        I = np.exp(- F) * I_initial
        return I # np.shape(I) = (n_y, n_x)
```

```
x=np.arange(-xsize/2,xsize/2)*pxl

T=2.*np.sqrt(Radius**2.-x**2.)
T=np.nan_to_num(T)
T=gaussian_filter(T, sigma=2)
ones_y=np.ones(ysize)
#Expand 1D to 2D with outer product
T=np.outer(ones_y,T)
###im = plt.imshow(T)
phase1=-k1*delta1*T
Int1=np.exp(-2.*k1*beta1*T)
```

The question is 'will my method give better results in phase retrieval?'

Week 9 (20/09/21 –26/09/21)

Aims:

1. Create simulations of phase contrast imaging for different targets

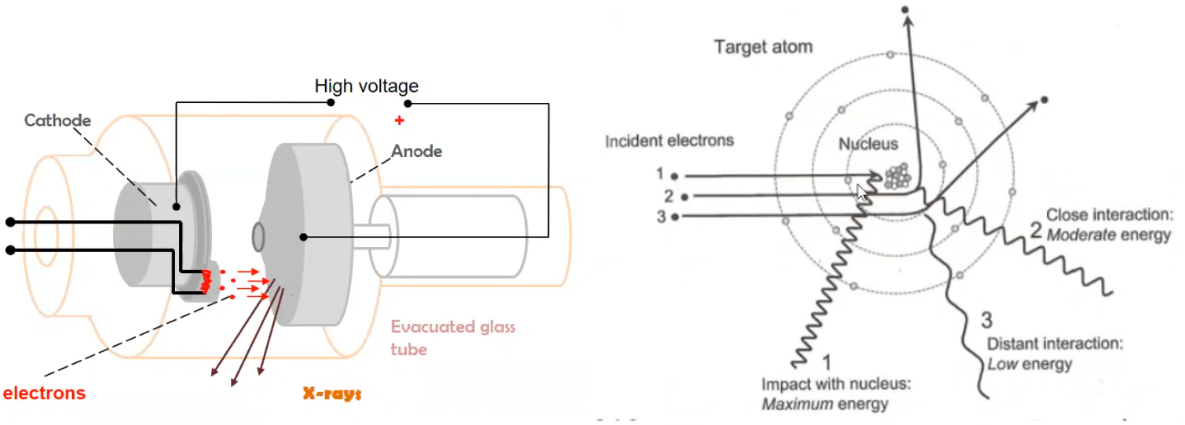
Tasks:

1. Meetings on **Monday 20/09** and **Wednesday 22/09**
    a. ~~Group: **2 pm**~~
    b. ~~one-on-one: **11 am**~~
2. **Read PyQt GUI book**
3. ~~**Edit progress report wk 9:**~~
    a. ~~**Submit 23rd September**~~

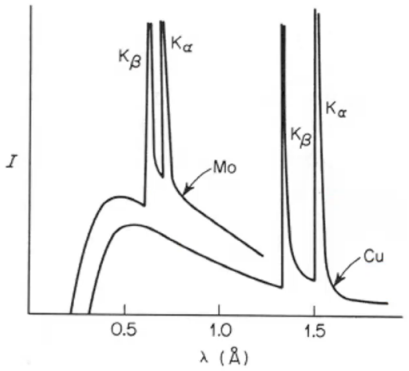
20/09/21

X-ray characteristic radiation

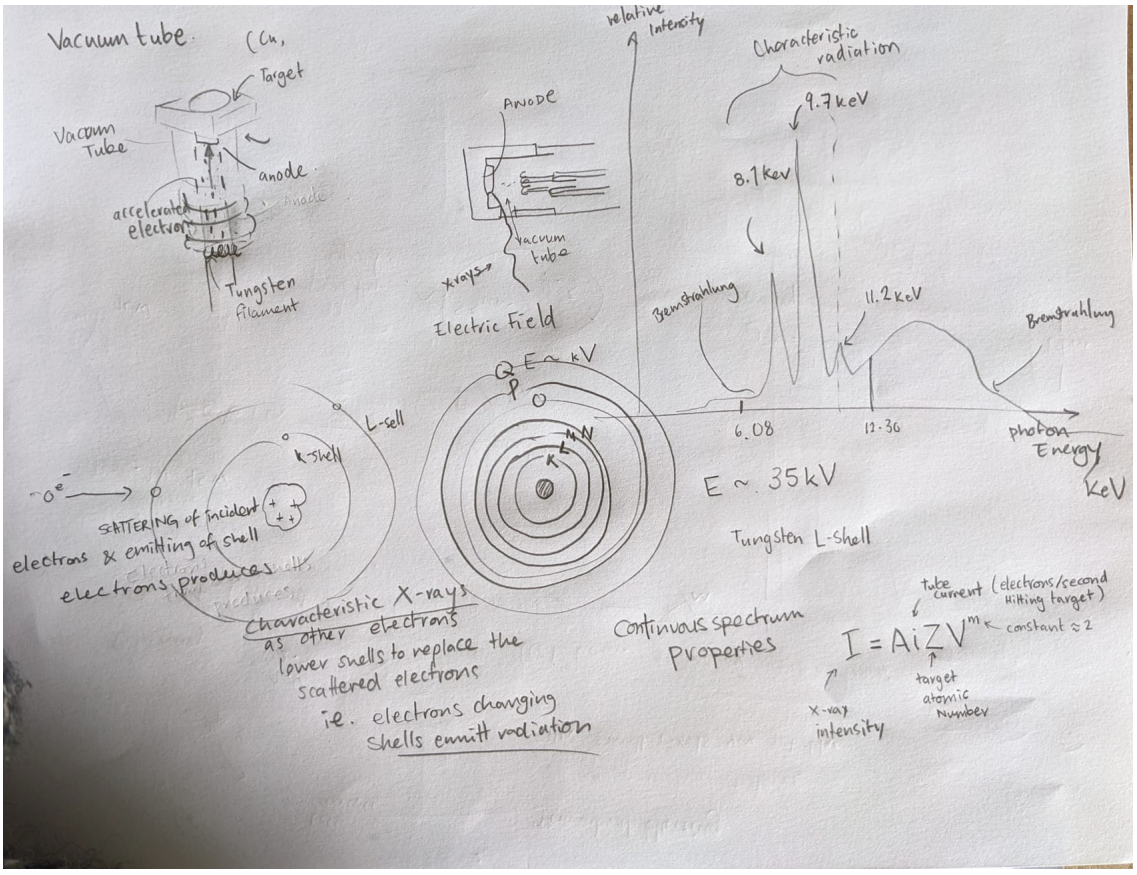
X-rays <-> High energy (small wavelength) ~ (10nm - 0.01nm)



High energy electrons against a metal plate generate X-ray scattering



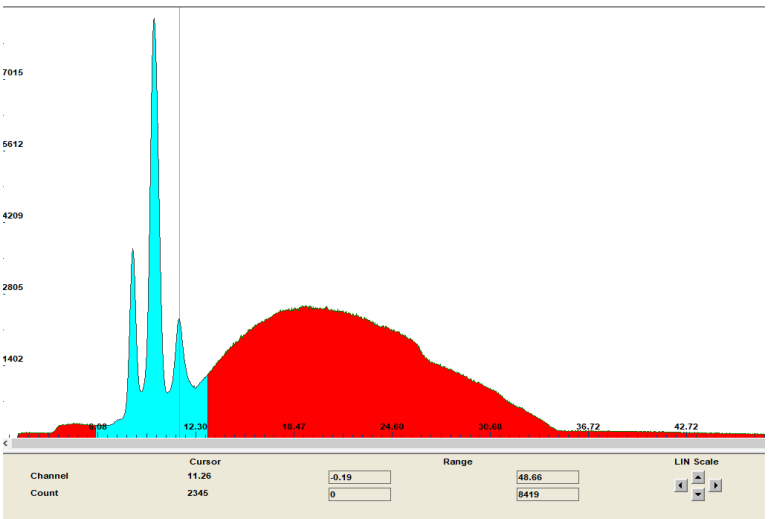
Typical X-ray spectrum

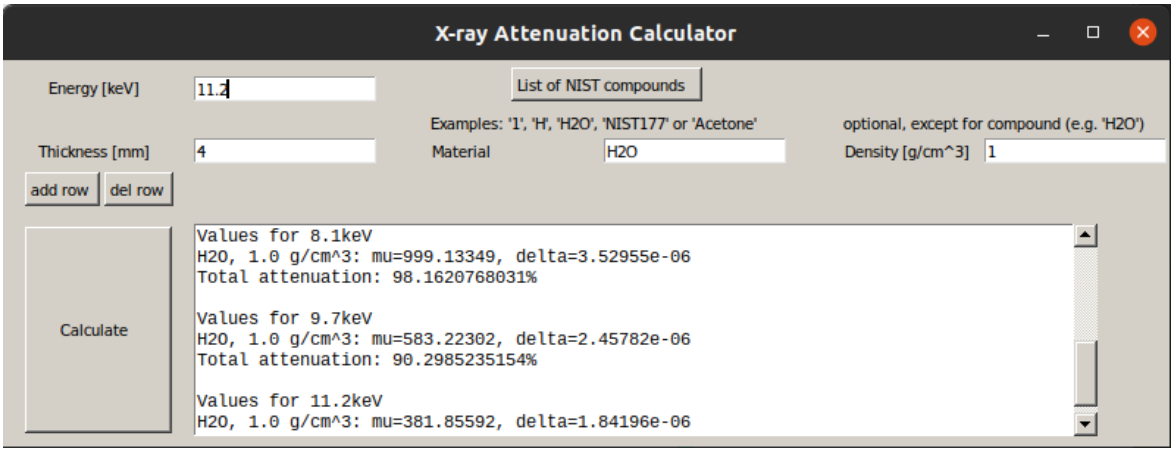


21/09/21

Tungsten vs Silver

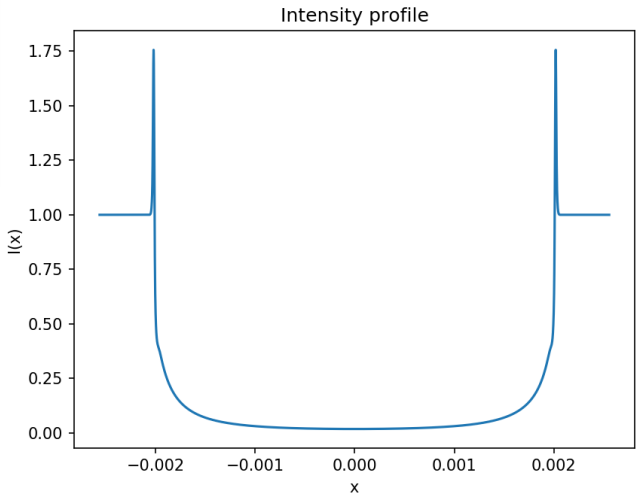
MK: "Here is a snapshot of one taken at 35 kV (the standard voltage we use) of the X-ray spectra of Tungsten measured in the lab. You can see 3 big characteristic radiation peaks from the tungsten L-shell at energies of ***~8.1****,* ***9.7*** *and* ***11.2 keV****. "*

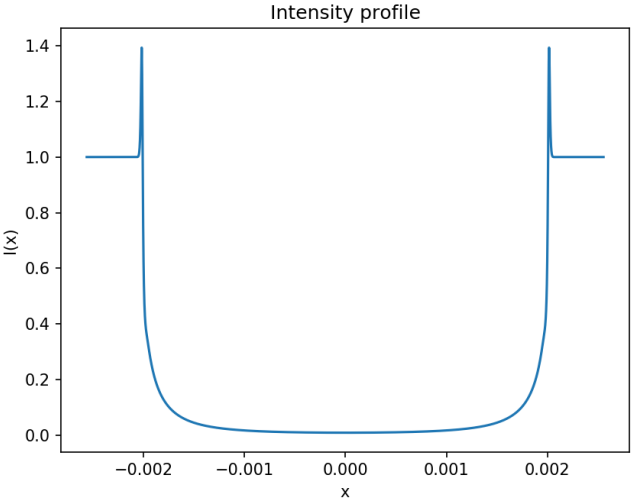
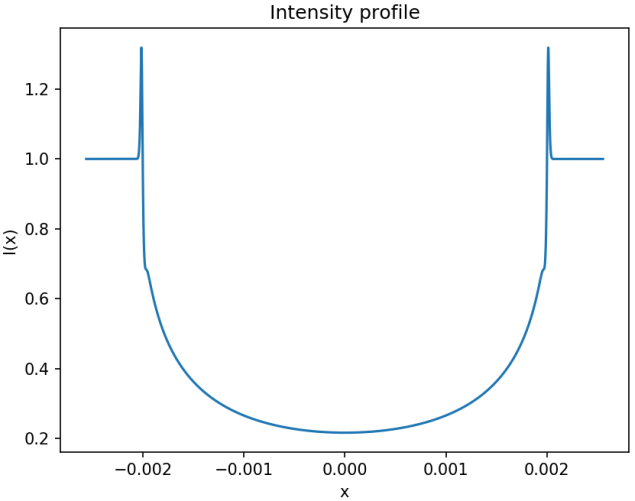


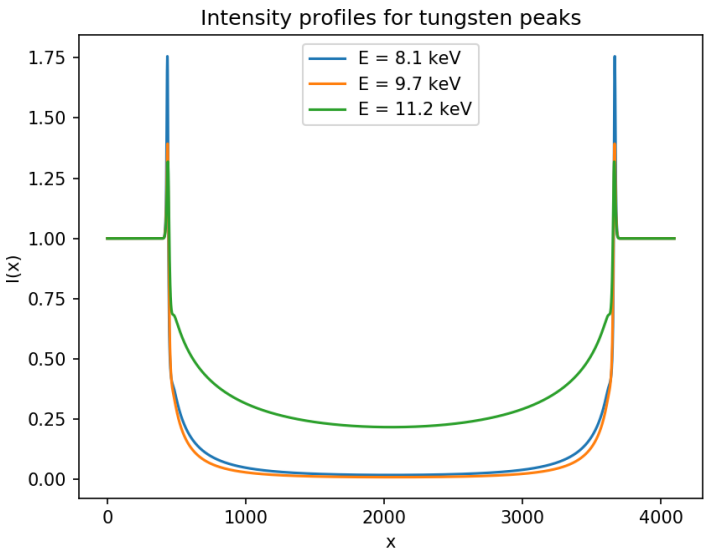


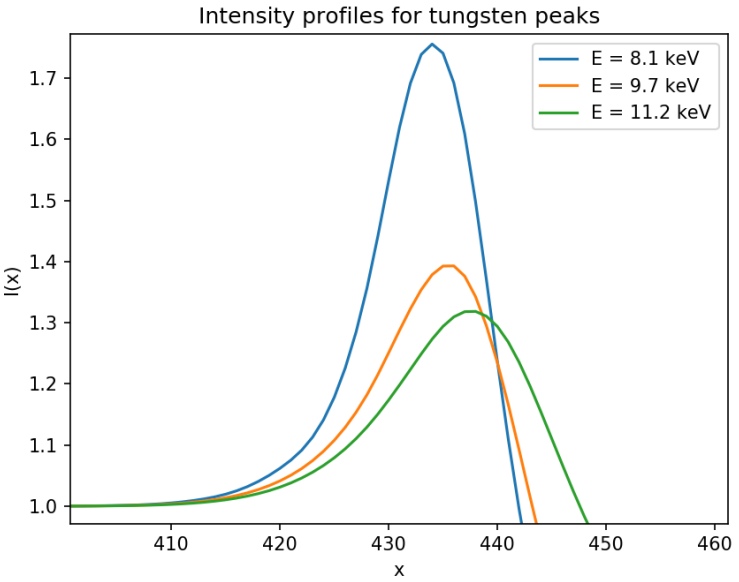
TESTING ANGULAR SPECTRUM
Tungsten

R = 2 * mm
# x-array parameters
n = 1024
n_x = n
x_max = (n_x / 2) * 5 * um
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
delta_x = x[1] - x[0]
# # y-array parameters
n_y = n
y_max = (n_y / 2) * 5 * um
y = np.linspace(-y_max, y_max, n_y, endpoint=False)
delta_y = y[1] - y[0]
y = y.reshape(n_y, 1)
z_final = 2.5 * m
supersample = 4
I = zoom(I, 4.0, order=3)
x = zoom(x, 4.0, order=3)

supersample = 3
 	
# Parameters from X-ray attenuation calculator 	
E1 = 8.1 # keV
λ1 = h * c / (E1 * 1000 * const.eV)
k1 = 2 * np.pi / λ1 # x-rays wavenumber1
# Material = water, density = 1 g/cm**3 	
δ1 = 352.955 * nm
μ1 = 999.13349 # per m
β1 = μ1 / (2 * k1)
	

E1 = 9.7 # keV **δ1 = 245.782 * nm** **μ1 = 583.22302 # per m**	
E1 = 11.2 # keV **δ1 = 184.196 * nm** **μ1 = 381.85592 # per m**	





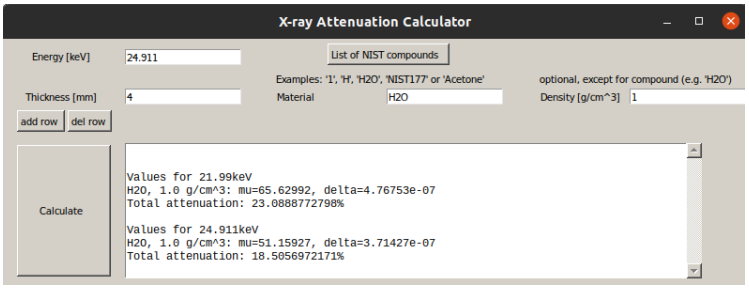
Silver

X-ray Transition Energies for silver (Deslattes et al.)

Element: Silver
Transitions: All
Energy Range: 20000 eV - 40000 eV
Sorted by Energy

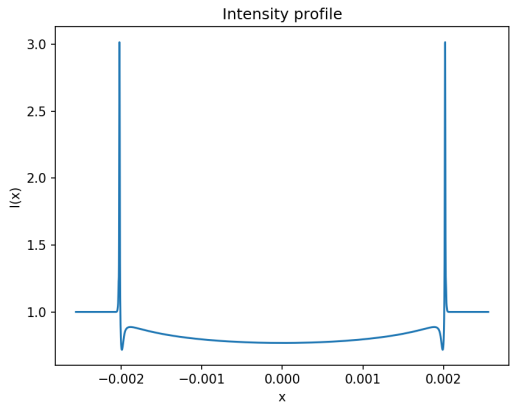
Download Formatted Text
Tab Delimited Fixed Column

Transition	Theoretical Energy (eV)	Experimental Energy (eV) Direct	Combined	Vapor	Blend	Ref.
KL_1	21 709.4(12)					
KL_2	21 990.67(72)	21 990.30(10)				5_d
KL_3	22 162.99(66)	22 162.917(30)				5_d
KM_1	24 797.4(12)					
KM_2	24 912.8(15)	24 911.54(30)				$1,5_d$
KM_3	24 943.1(13)	24 942.42(30)				$1,5_d$
KM_4	25 141.7(11)	25 145.5(15)			$KM_{4,5}$	1_d
KM_5	25 147.6(10)	25 145.5(15)			$KM_{4,5}$	1_d
KN_1	25 421.1(35)					
KN_2	25 455.6(29)	25 456.71(31)			$KN_{2,3}$	1_d
KN_3	25 457.8(20)	25 456.71(31)			$KN_{2,3}$	1_d
KN_4	25 509.99(91)	25 511.8(31)			$KN_{4,5}$	1_d
KN_5	25 510.83(99)	25 511.8(31)			$KN_{4,5}$	1_d
K edge	25 523.71(39)	25 515.59(30)	25 515.51(48)			13_d



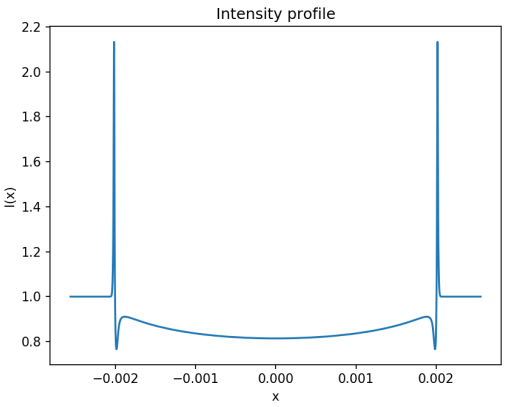
E1 = 21.99 # keV

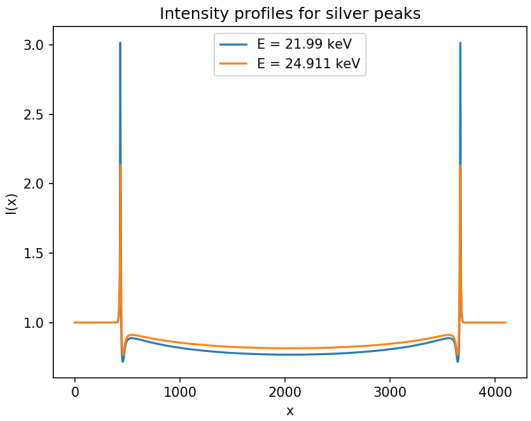
δ1 = 476.753 * nm
μ1 = 65.62992 # per m

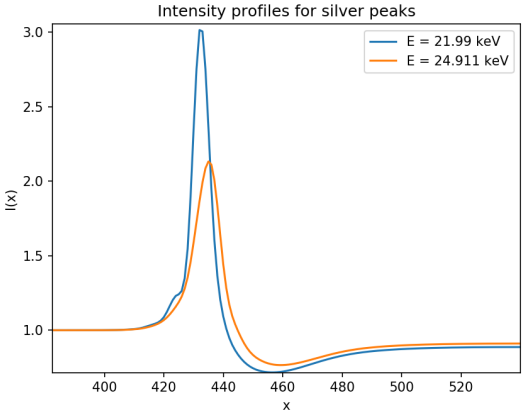


E1 = 24.911 # keV

δ1 = 37.1427 * um
μ1 = 51.15927 # per m







24/09/21

MK's latest comments

I had further thoughts for your simulations. We discussed that the Angular Spectrum (AS) is the 'gold standard' to compare against. As you know, the AS also shows that the peak position will shift with distance even though the TIE peaks never budge - at least with the finite difference approximation. If your Runge-Kutta approach enables us to see shifts of the peaks with distance, this will be even better proof than the peak amplitude that it is a better approach than the standard finite difference approach. I look forward to seeing if this is the case :)

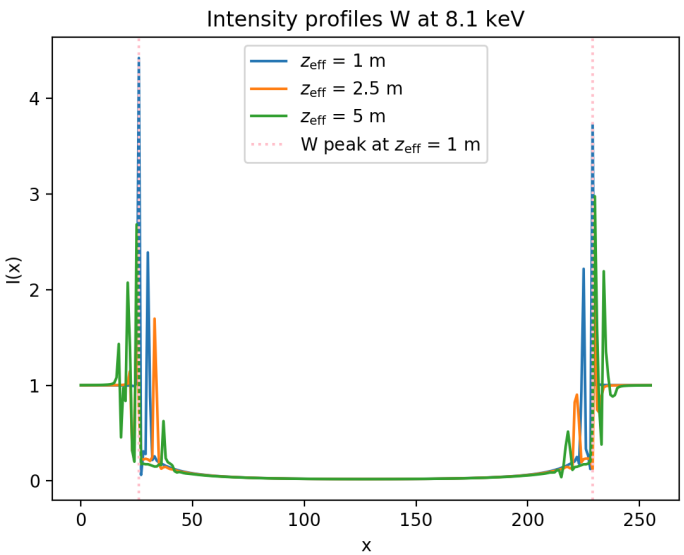
Corrected Tungsten

Here I am testing the Tungsten and silver peaks again because I noticed that I had written the delta values with the wrong order of magnitude, so I copied directly the deltas from the attenuation calculator… This change yields lots of ringing artefacts :(I guess it's probably right??

I use: I = xri.sim.propAS(δT, βT, E, z_final, delta_x, supersample=3) for all the data sets

I've tried increasing ss to decrease ringing but my RAM can't deal

Testing different propagation distance with the same energy peak (W)



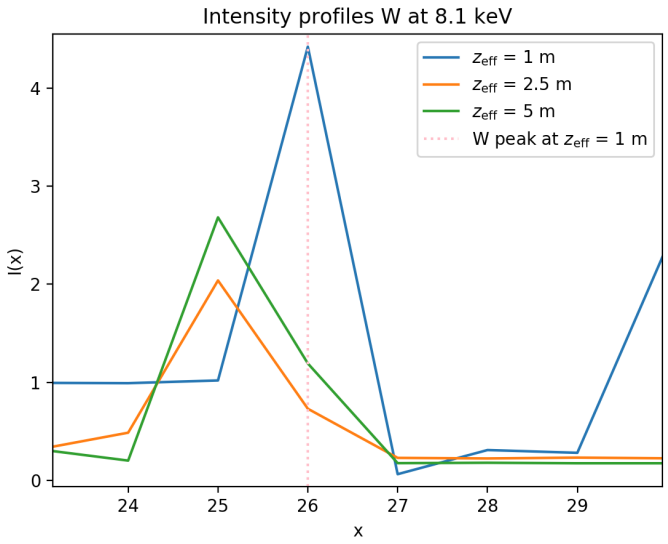
Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

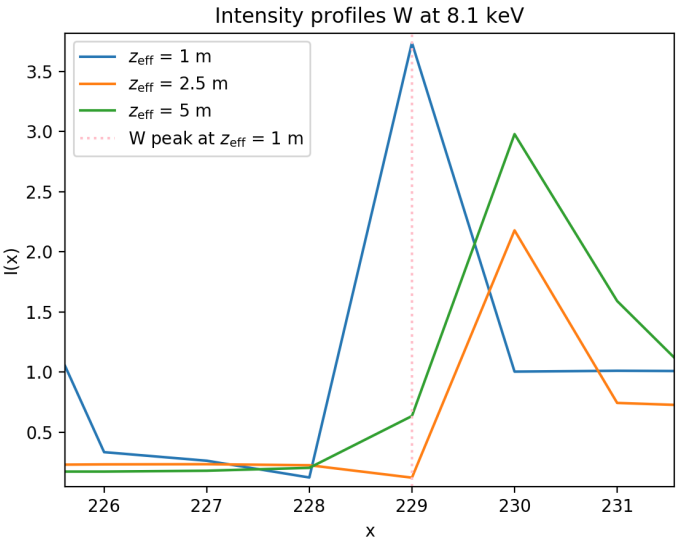
The ringing is very strong in these plots and the asymmetry of the phase contrast fringes for all energies used in this plot is visible.

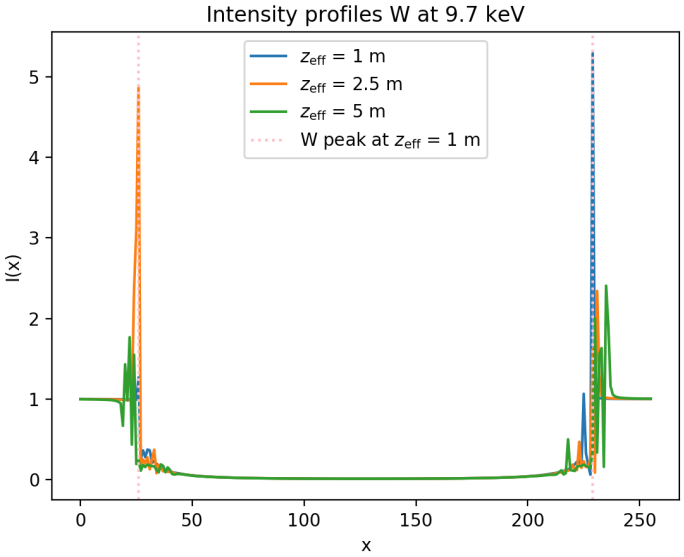
In the 2 *enhanced* figures below:
I can observe that for z = 2.5m and z = 5m the brightest fringes at this energy have moved outward from the dotted pink lines, but these fringes remain apparently aligned with each other.

What's with the asymmetry in the left and right peak heights?

Left peaks *enhance*	**Right peaks *enhance***





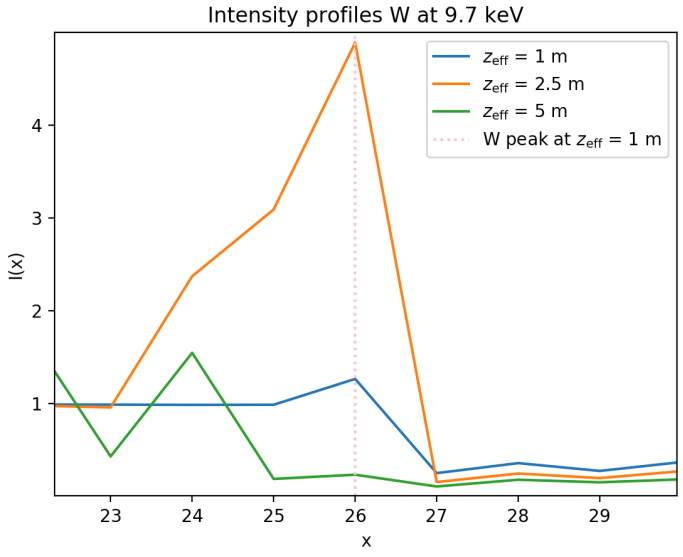


Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

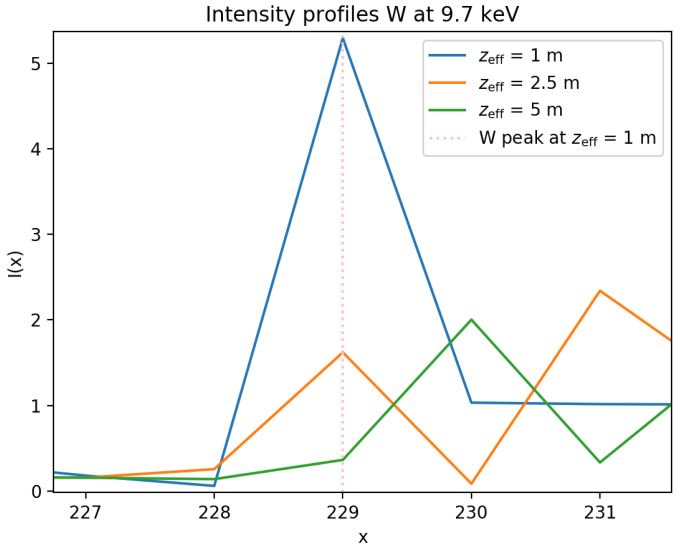
The ringing is very strong in these plots and the asymmetry of the phase contrast fringes for all energies used in this plot is visible.

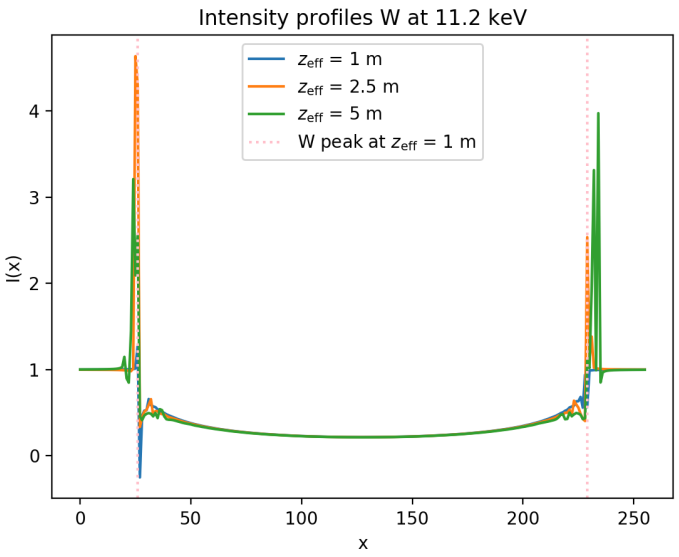
In the 2 *enhanced* figures below:
I can observe that for z = 2.5m and z = 5m it is more difficult to identify the phase contrast fringes because of the ringing.

Right peaks *enhance*



Right peaks *enhance*





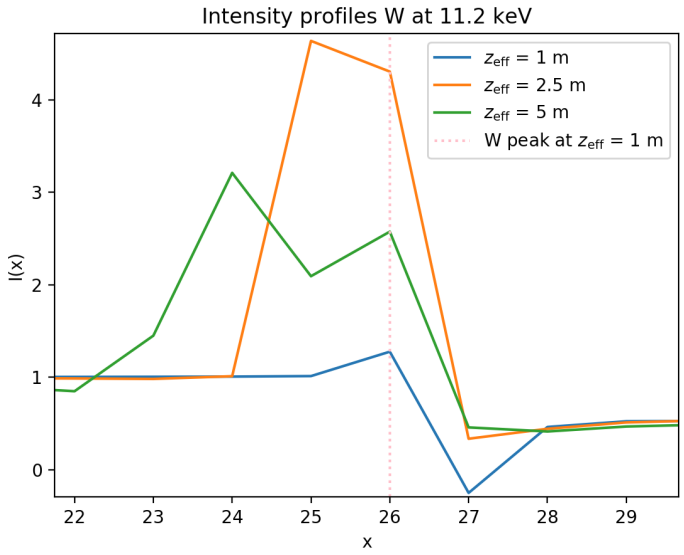
Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

The ringing is very strong in these plots and the asymmetry of the phase contrast fringes for all energies used in this plot is visible.

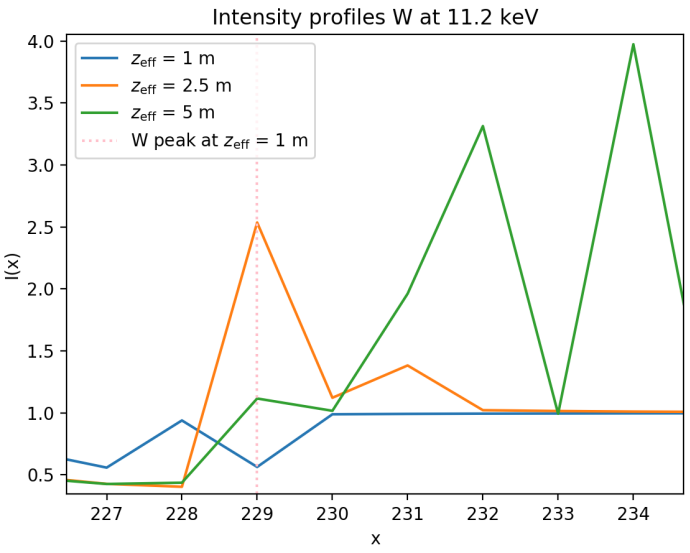
In the 2 *enhanced* figures below:
I can observe that for z = 2.5m and z = 5m it is more difficult to identify the phase contrast fringes because of the ringing.

For the 5m propagation distance I can see that the fringe's height is consistently smaller when compared to the other two distances. The 1m and 2.5 m distances seem to swap the "highest" peak in the left and right enhanced figures.

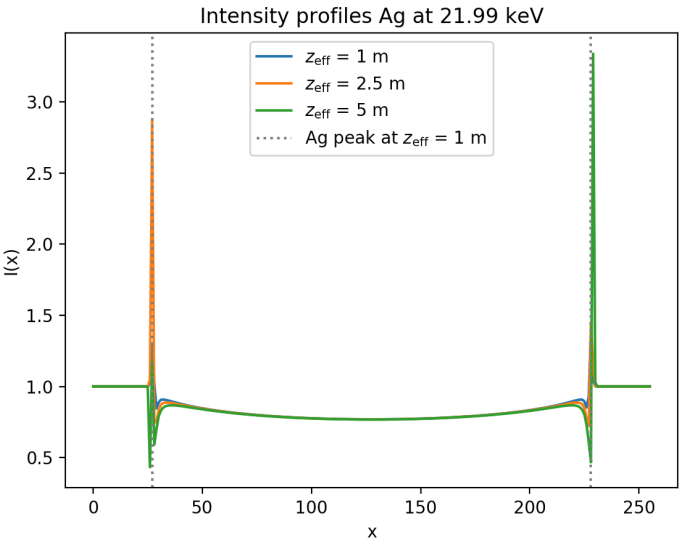
Left peaks *enhance*



Right peaks *enhance*



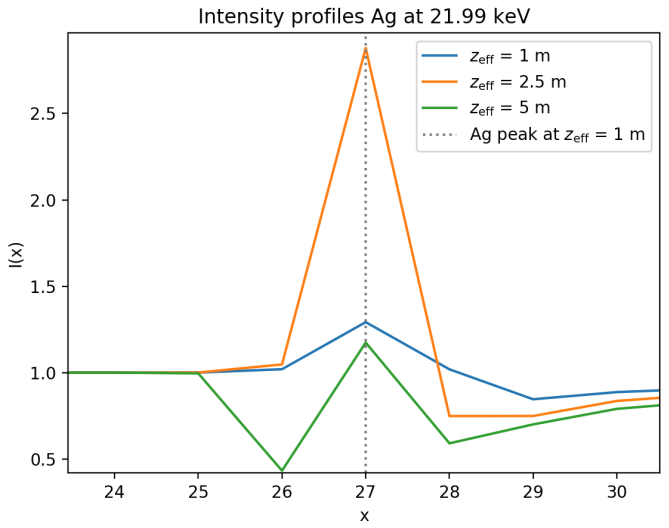
Testing different propagation distance with the same energy peak (Ag)

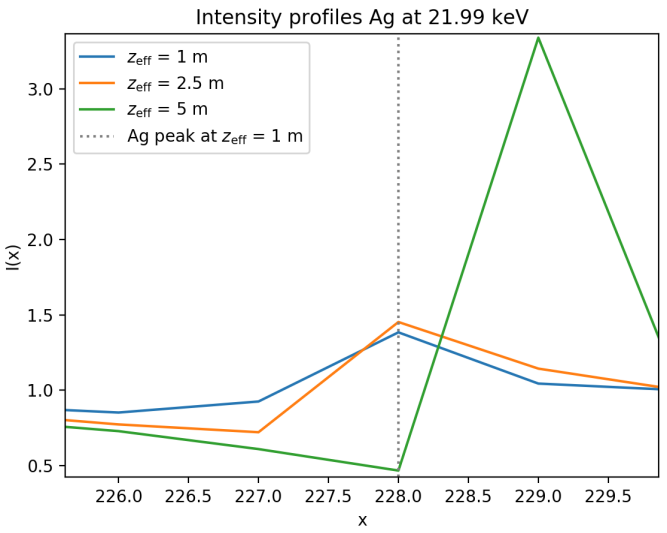


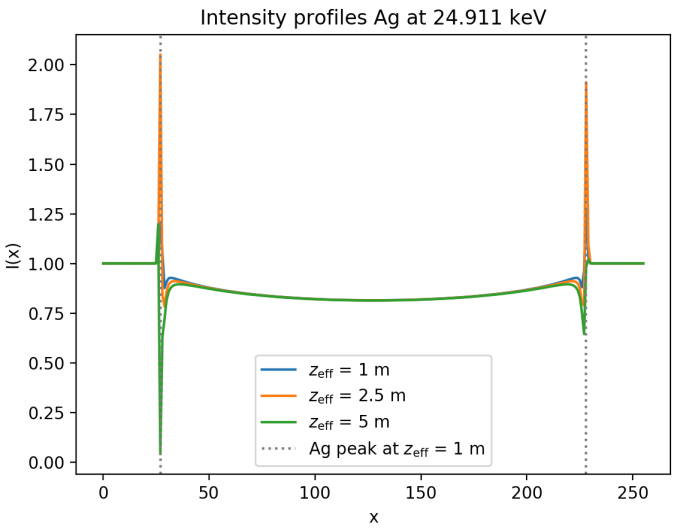
UGH SO UGLY

Left peaks *enhance*

Right peaks *enhance*



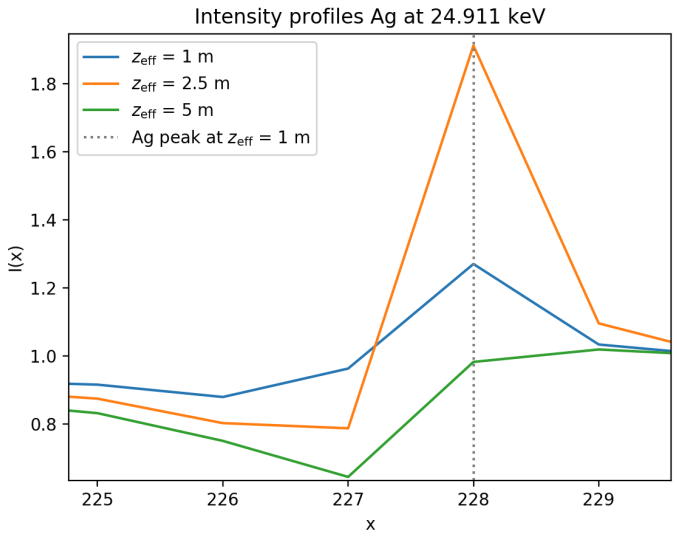




OMG GROSS

Left peaks *enhance*

Right peaks *enhance*



IF *the Angular Spectrum (AS) is the 'gold standard' to compare against*… I suppose he means AS is the **analytical** gold standard.

25/09/21

Test TIE+RK

Material = water, density = 1 g/cm**3
D = 4 * mm

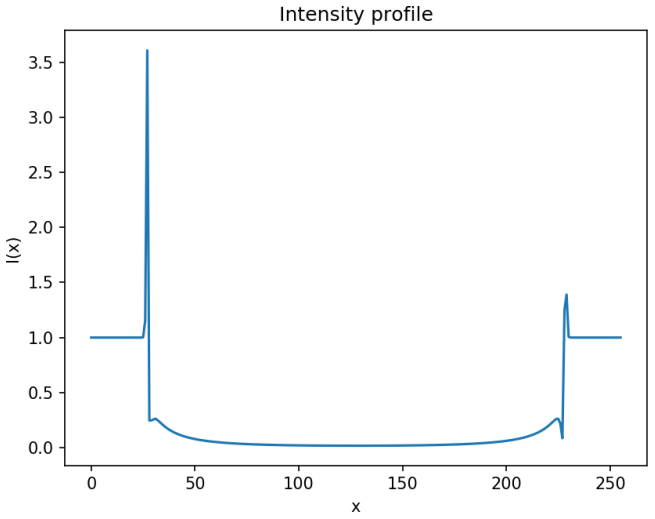
E = 8.1 * keV
δ = 3.52955e-06
μ = 999.13349 # per m
β = μ1 / (2 * k)

n_x = **1024**
n_y = **512**

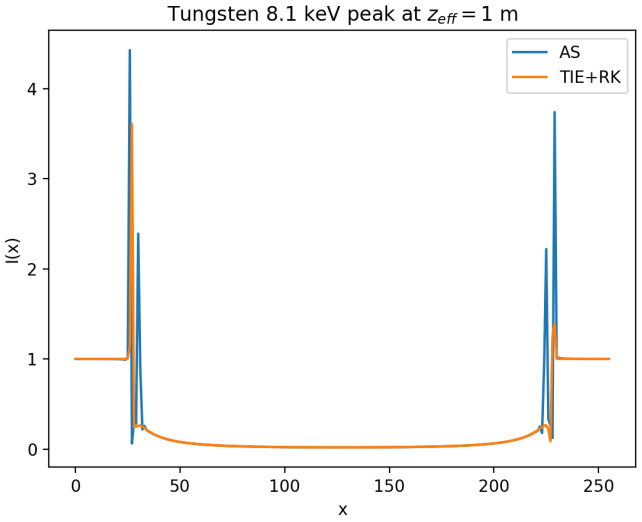
x_max = (**n_x** / 2) * 5 * um
x = np.linspace(-x_max, x_max, **n_x**, endpoint=False)
y_max = (**n_y** / 2) * 5 * um
y = np.linspace(-y_max, y_max, **n_y**, endpoint=False).reshape(**n_y**, 1)

Fourth order Runge-Kutta
z_final = 1 * m # propagation distance
delta_z = 10 * mm # (n_z = 100)

zoom(I, 0.25, order=3)



Comparing TIE+RK vs AS



1. Instability appears more prevalent in **AS** method
2. The peaks in **AS** are higher
3. The asymmetry in both methods is probably due to the **zoom** function… Maybe I don't have the correct resolution in the plot?
   SCREW RESAMPLING FOR THE TIME BEING

NO ZOOM

Material = water, density = 1 g/cm**3
D = 4 * mm

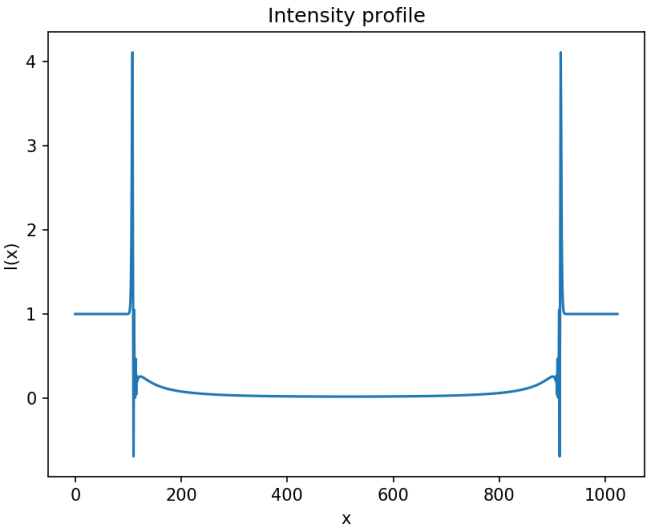
E = 8.1 * keV
δ = 3.52955e-06
μ = 999.13349 # per m
β = μ1 / (2 * k)

n_x = **1024**
n_y = **512**

x_max = (**n_x** / 2) * 5 * um
x = np.linspace(-x_max, x_max, **n_x**, endpoint=False)
y_max = (**n_y** / 2) * 5 * um
y = np.linspace(-y_max, y_max, **n_y**, endpoint=False).reshape(**n_y**, 1)

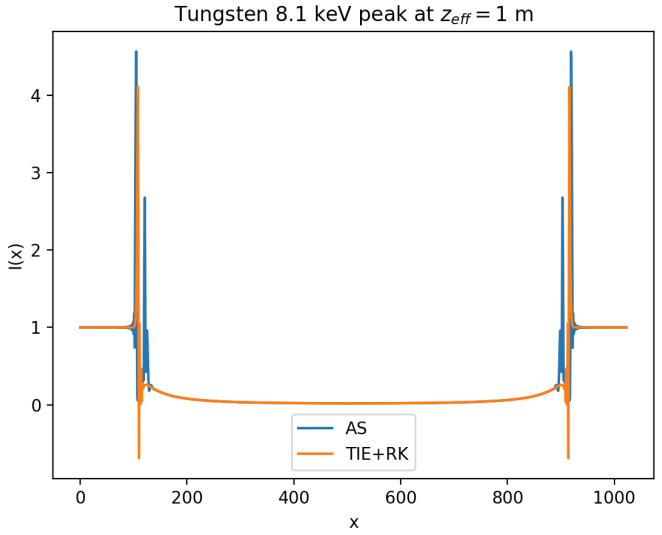
Fourth order Runge-Kutta
z_final = 1 * m # propagation distance
delta_z = 1 * mm # (n_z = 1000)

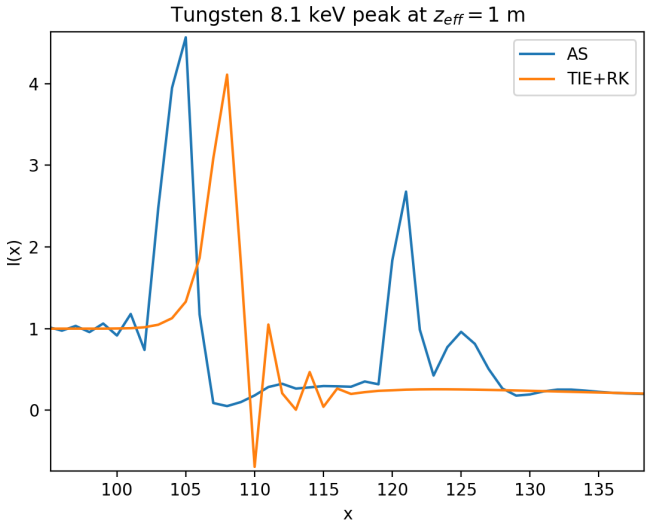
NO ZOOM

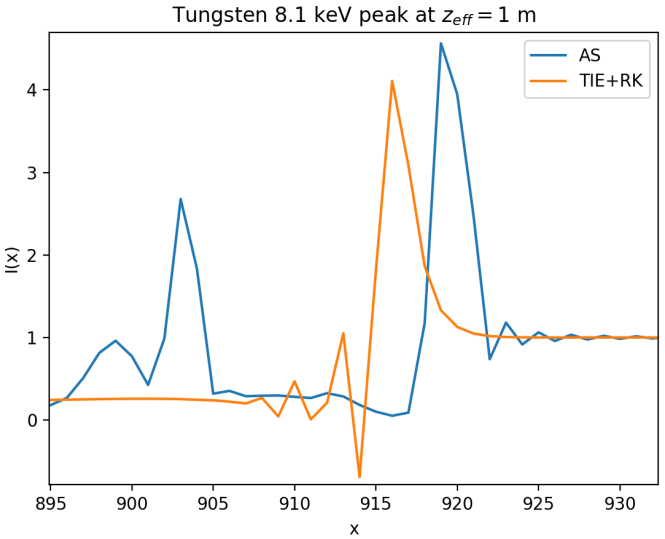


Comparing TIE+RK vs AS

1. Instability is higher in **AS**
2. Peaks are still higher in **AS**







Increase the sigma in the gaussian filter

Material = water, density = 1 g/cm**3
D = 4 * mm

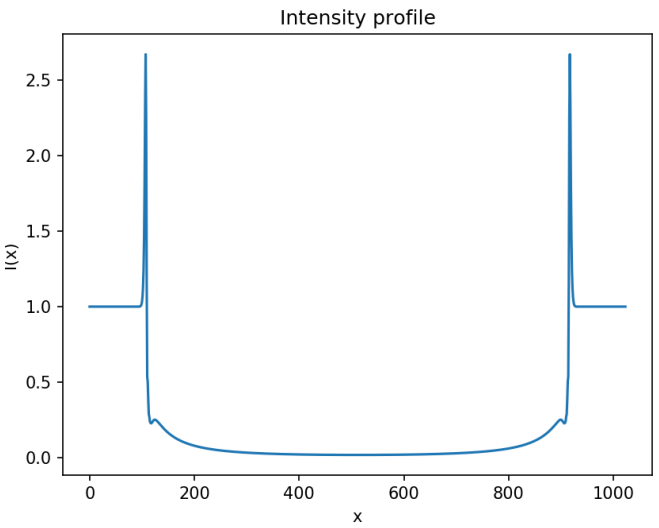
E = 8.1 * keV
δ = 3.52955e-06
μ = 999.13349 # per m
β = μ1 / (2 * k)

n_x = **1024**
n_y = **512**

x_max = (**n_x** / 2) * 5 * um
x = np.linspace(-x_max, x_max, **n_x**, endpoint=False)
y_max = (**n_y** / 2) * 5 * um
y = np.linspace(-y_max, y_max, **n_y**, endpoint=False).reshape(**n_y**, 1)

Fourth order Runge-Kutta
z_final = 1 * m # propagation distance
delta_z = 1 * mm # (n_z = 1000)

T = gaussian_filter(T, sigma=4)
NO ZOOM ←fixes asymmetry as expected

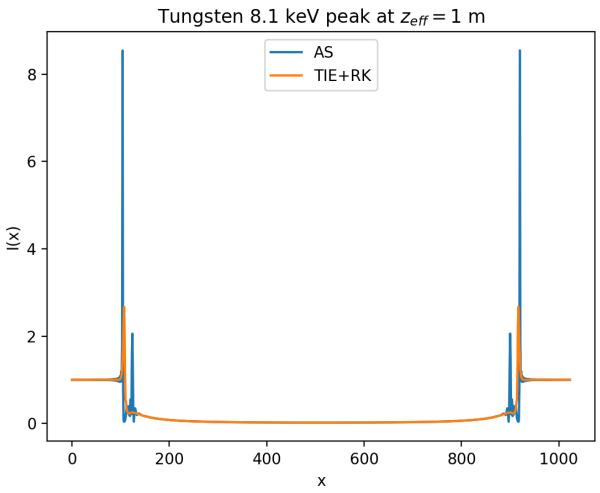


Comparing TIE+RK vs AS

Here I tested both **AS** and **TIE+RK** with
T = gaussian_filter(T, sigma=4)
NO ZOOM resampling

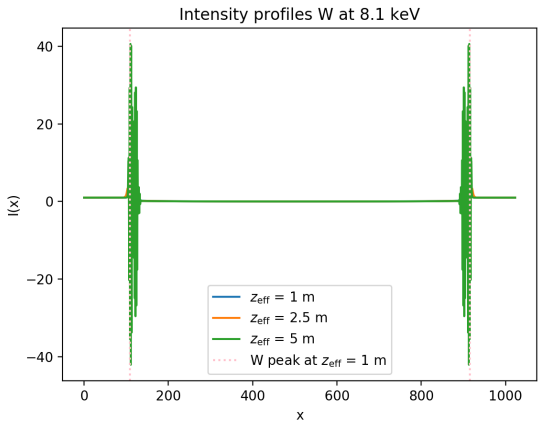
1. **AS** appears to have unphysically high fringes
2. Instability is higher in **AS**
3. **TIE+RK** appears completely smooth

I will retest the AS plots without the ZOOM resampling!!
And with gaussian_filter(T, sigma=4)

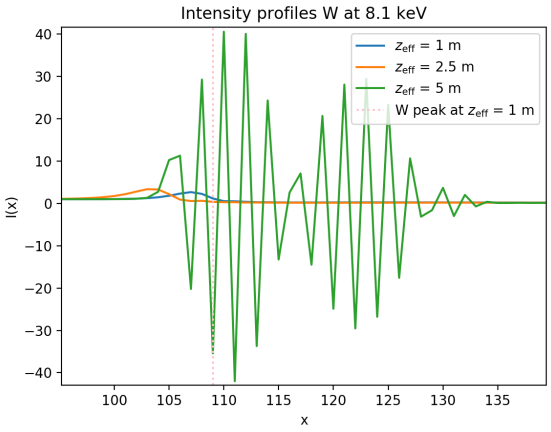


26/09/21

TUNGSTEN (W)



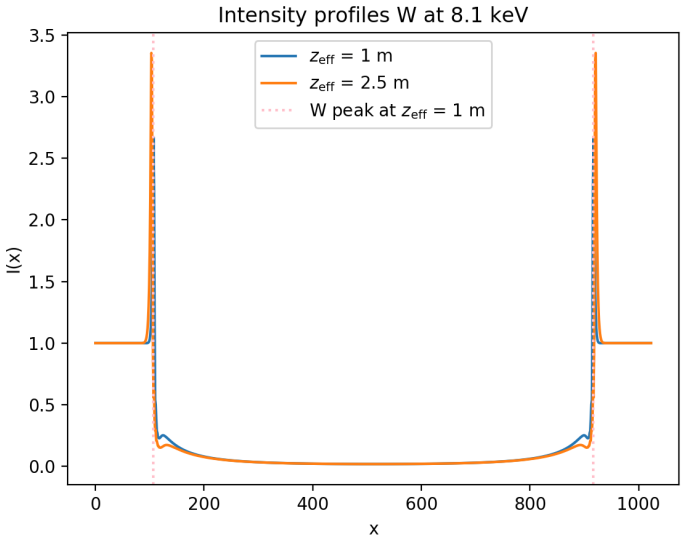
Left fringes *enhance!*



WOW wiggles!

My evaluation of the TIE with RK at the propagation distance of 5m is not very reliable...
I don't think that this green plot provides any useful info in this sim other than demonstrating that my z spatial step might be too large. I didn't get to simulate this yet but I'll try halving the size of the delta_z value I mentioned above.

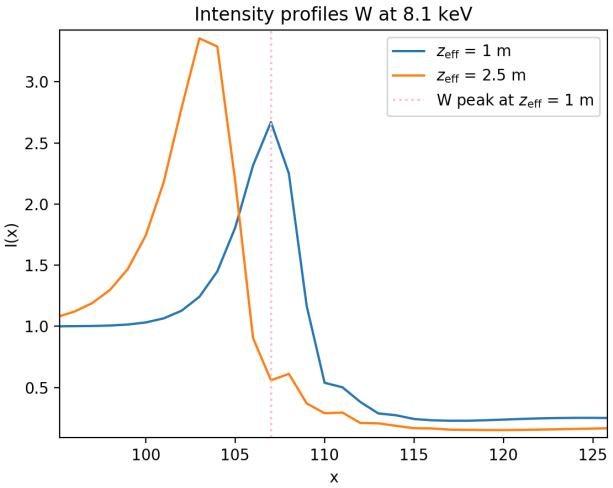
This instability is only occurring in the lower energies (i.e. the tungsten peaks), therefore, I will temporarily neglect the data from the z_eff = 5m for the tungsten energies.

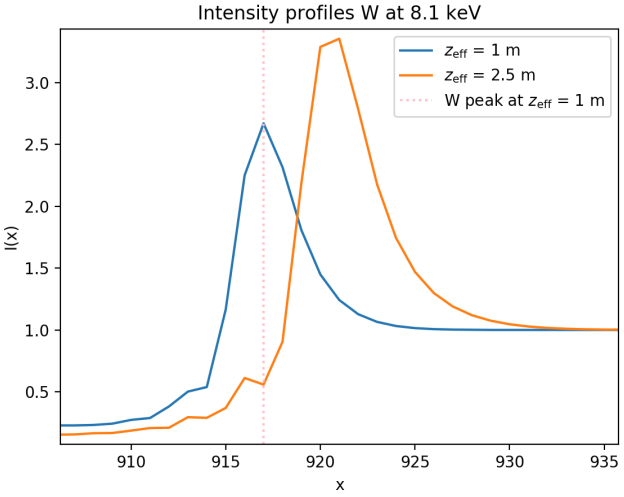


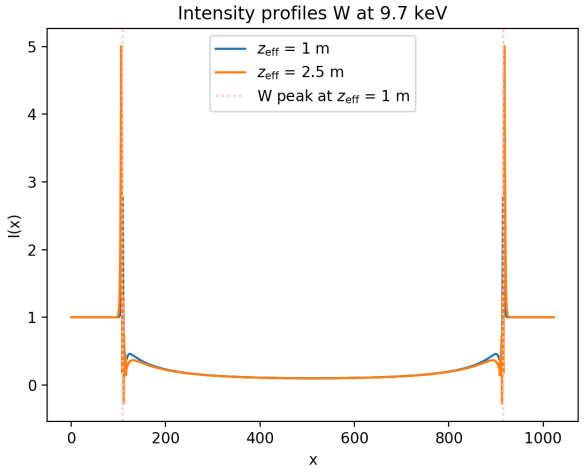
Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

I have not included z_eff = 5m data, since the ringing is enormous and so it affects the display of the other plots. **I think the issue could be solved by decreasing the z step size for RK. I haven't tested that yet.**

In the 2 *enhanced* figures below:
I can observe that the brightest fringes for z_eff = 2.5m at this energy have moved outward from the dotted pink lines.





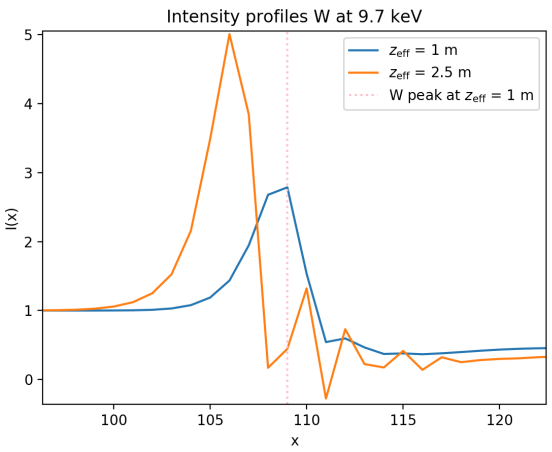


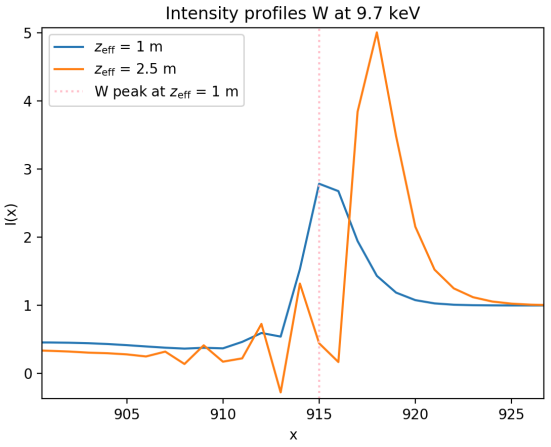
Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

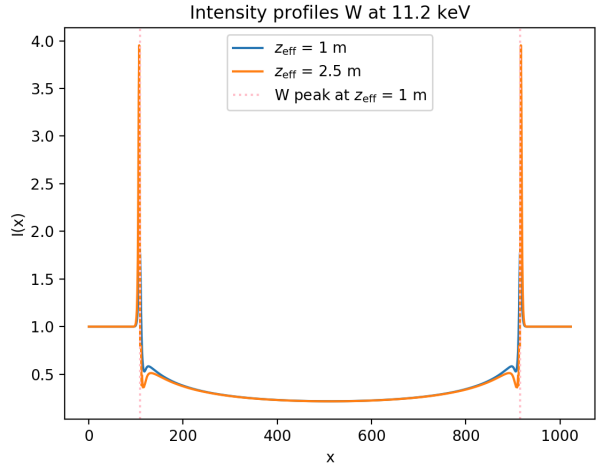
I have not included z_eff = 5m data, since the ringing is enormous and so it affects the display of the other plots. **I think the issue could be solved by decreasing the z step size for RK. I haven't tested that yet.**

Ringing is stronger than the data for the W - peak @ 8.1keV
In the case of z_eff = 2.5m data sets.

In the 2 *enhanced* figures below:
I can observe that the brightest fringes for z_eff = 2.5m at this energy have moved outward from the dotted pink lines.





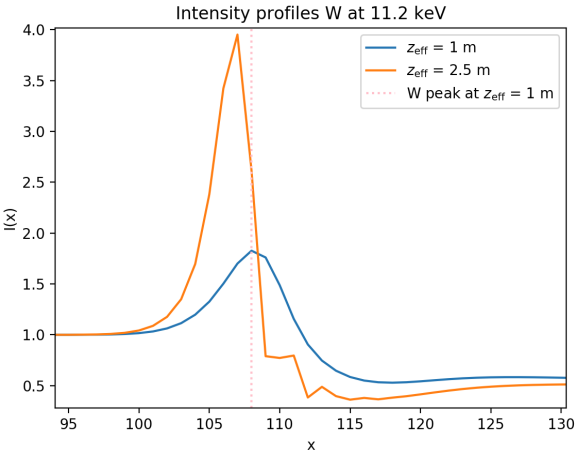


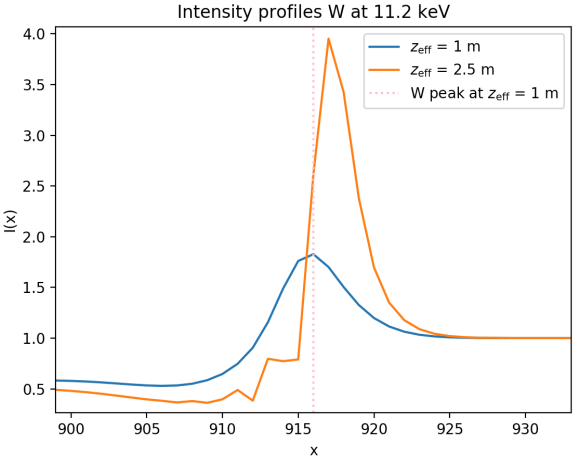
Here I have a vertical dotted pink line that marks the place where the contrast fringes appear to be highest for the propagation distance z = 1m

I have not included z_eff = 5m data, since the ringing is enormous and so it affects the display of the other plots. **I think the issue could be solved by decreasing the z step size for RK. I haven't tested that yet.**

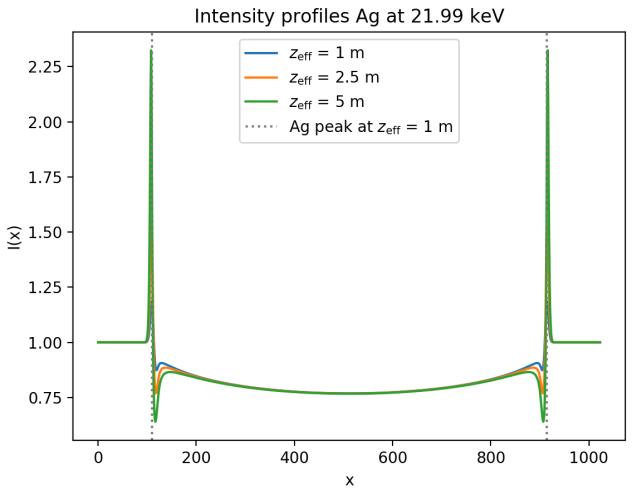
Ringing is stronger than the data for the W - peak @ 8.1keV
In the case of z_eff = 2.5m data sets.

In the 2 *enhanced* figures below:
I can observe that the brightest fringes for z_eff = 2.5m at this energy have moved outward from the dotted pink lines.



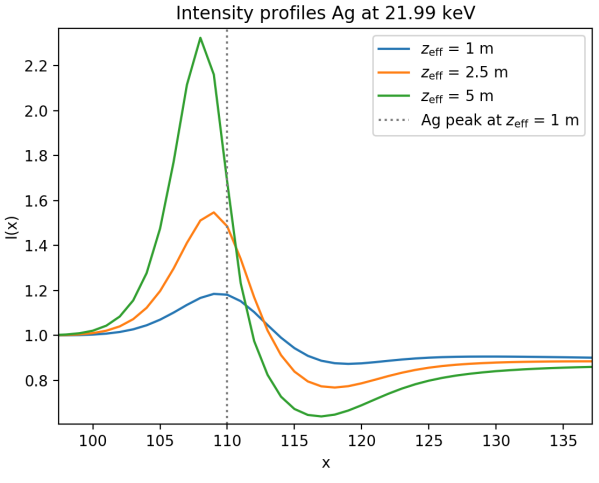


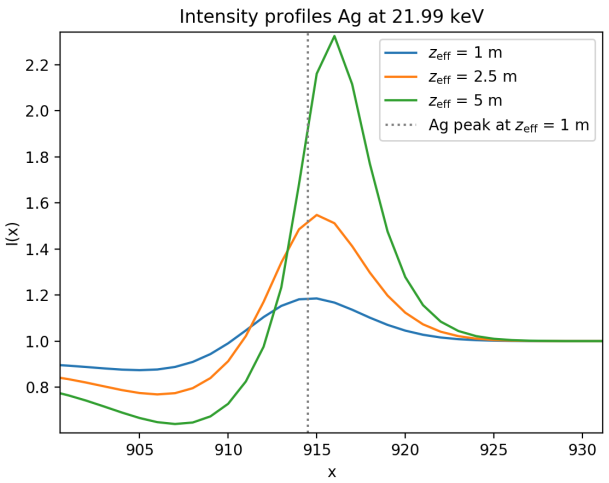
SILVER (Ag)

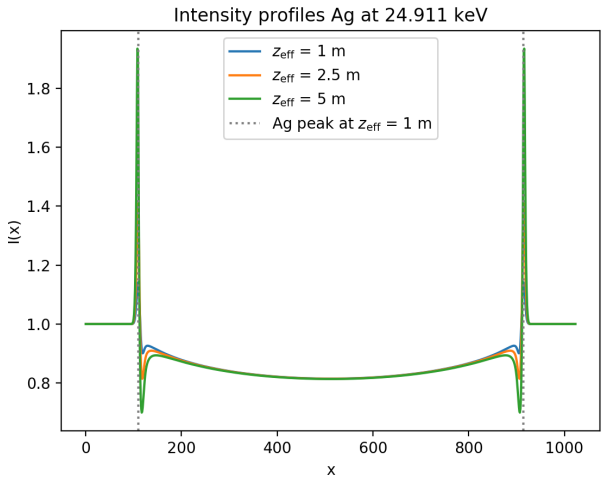


The fringes move outward as propagation distance increases

No ringing is visible!

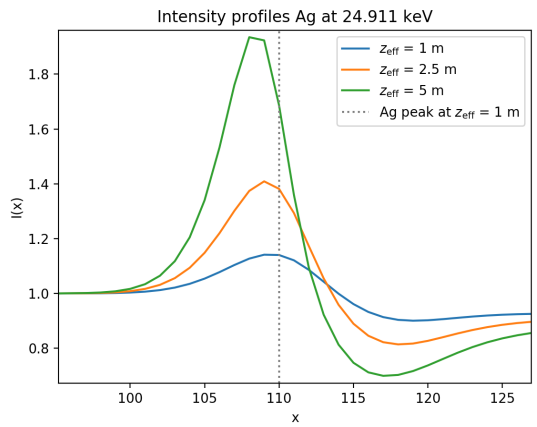


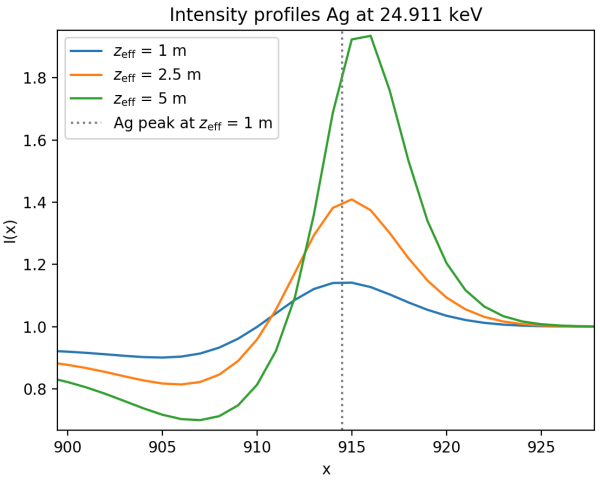




The fringes move outward as propagation distance increases

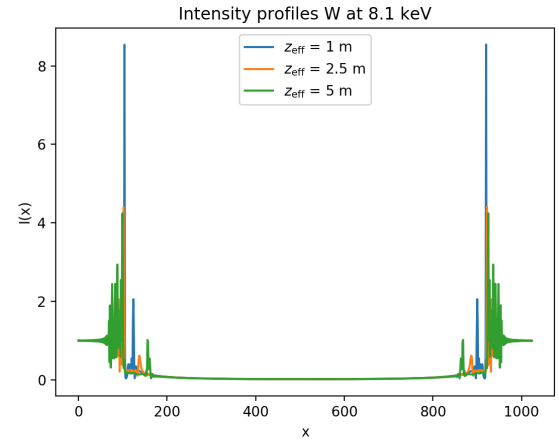
No ringing is visible!





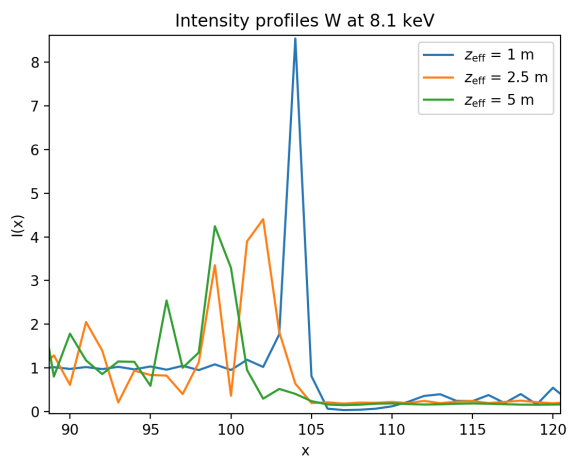
Testing different propagation distance with the same energy peak - using only xri's AS

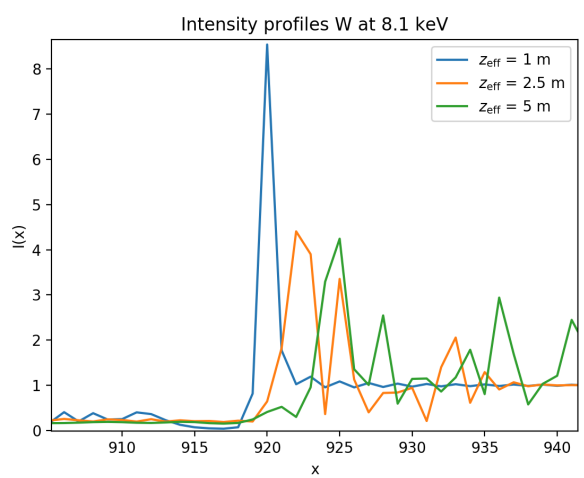
TUNGSTEN (W)

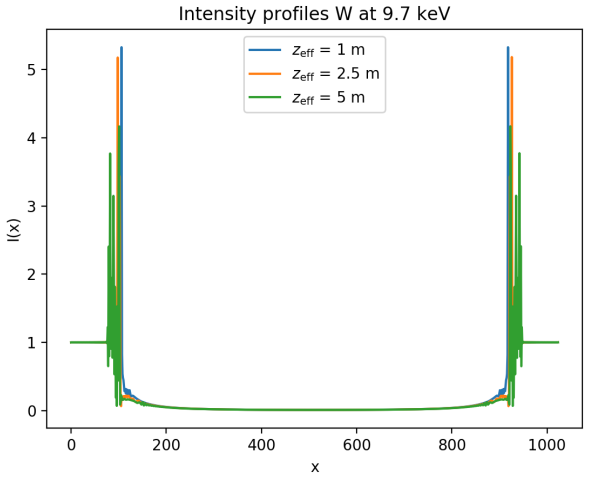


The ringing is very strong in these plots and the fringes are probably too tall to represent real physics.

In the 2 *enhanced* figures below:
I can observe that for z_eff = 2.5m and z_eff = 5m the apparent brightest fringes at this energy have moved outward as expected.

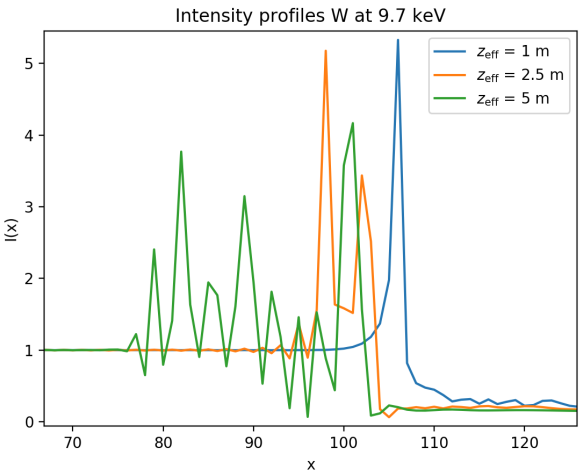


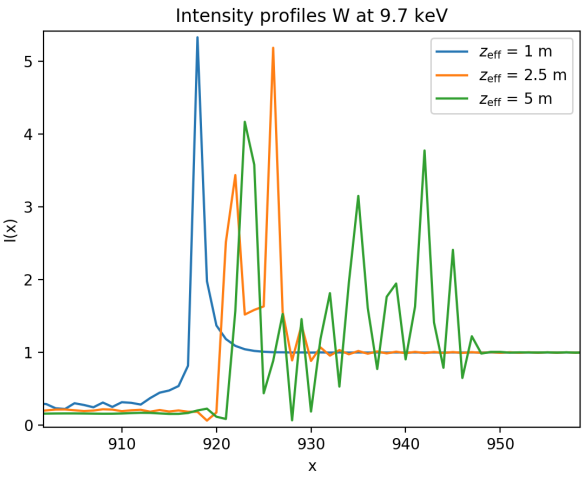


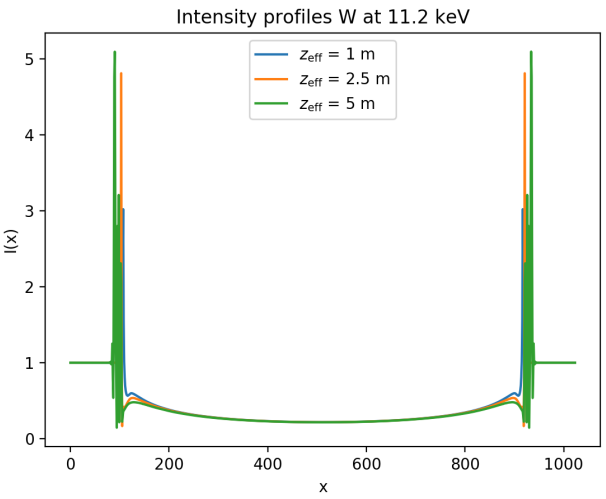


The ringing is stronger than in the previous energy test using the AS method.

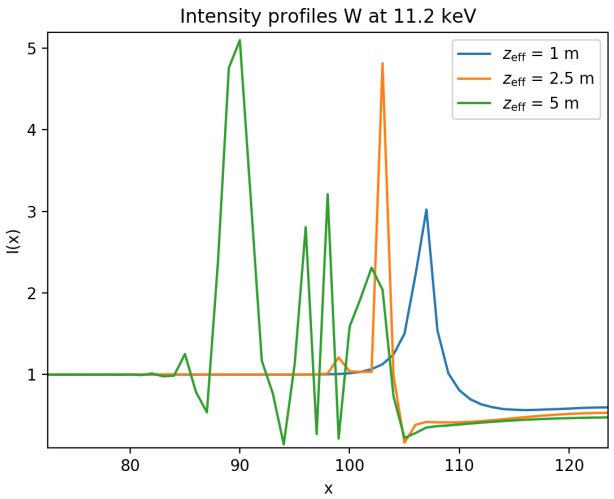
In the 2 *enhanced* figures below:
I can observe that for z_eff = 2.5m and z_eff = 5m the apparent brightest fringes at this energy have moved outward as expected.

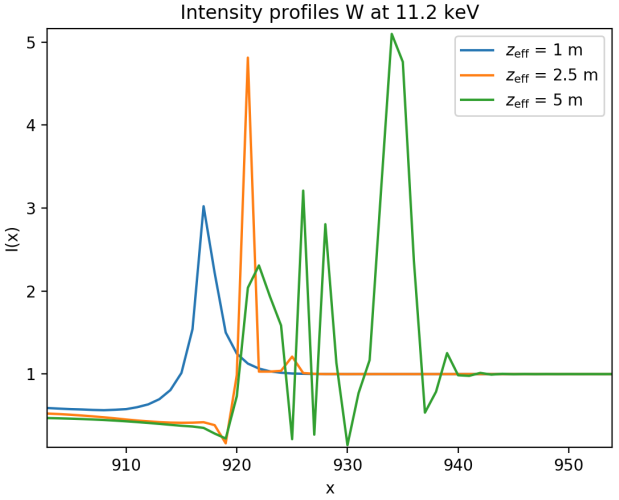




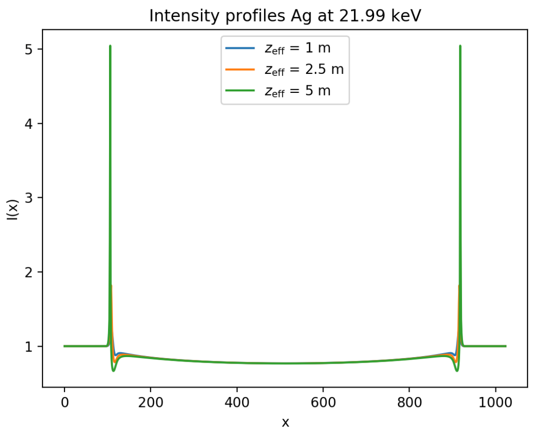


In the 2 *enhanced* figures below:
I can observe that for z_eff = 2.5m and z_eff = 5m the apparent brightest fringes at this energy have moved outward as expected. The ringing presents a challenge interpreting the data though.

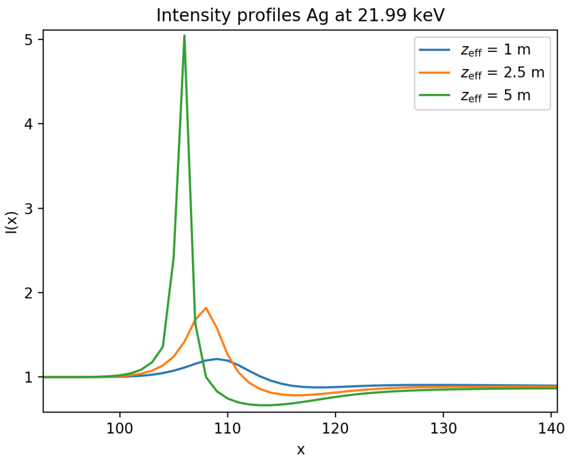


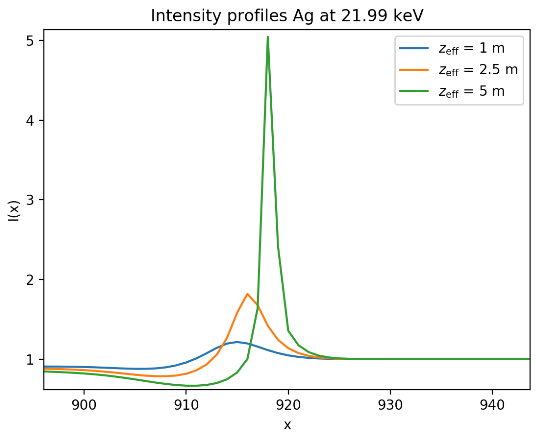


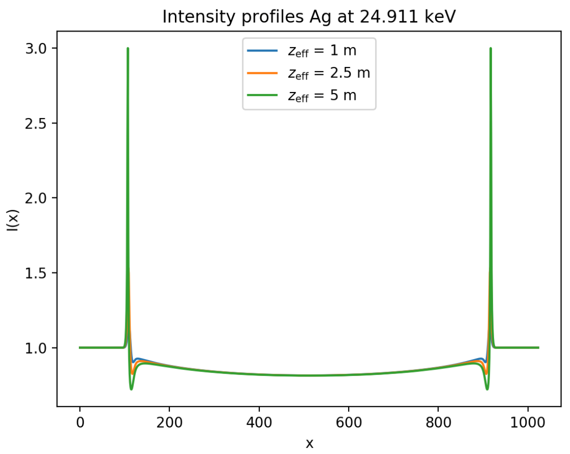
SILVER (Ag)

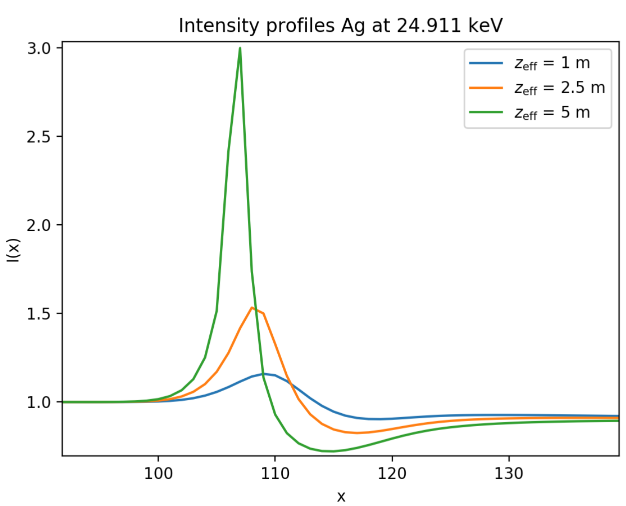


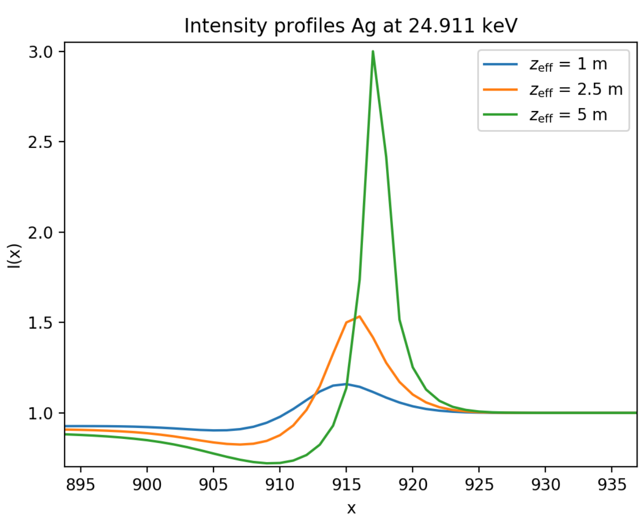
AS looks great here!
As expected the fringes move outward with distance. :)





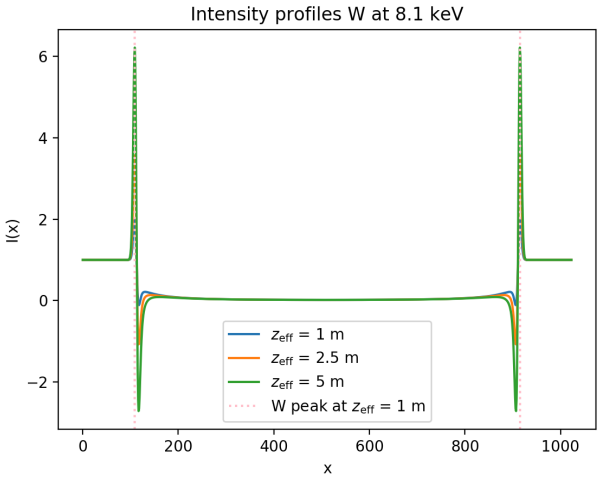






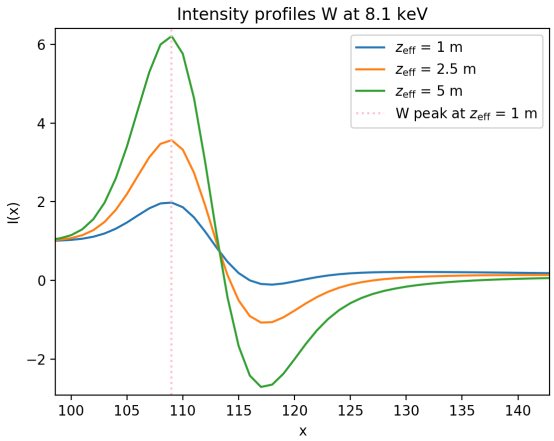
Testing different propagation distance with the same energy peak - using only xri's TIE

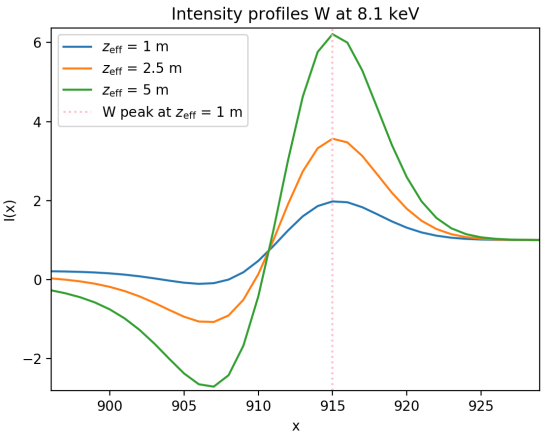
TUNGSTEN (W)

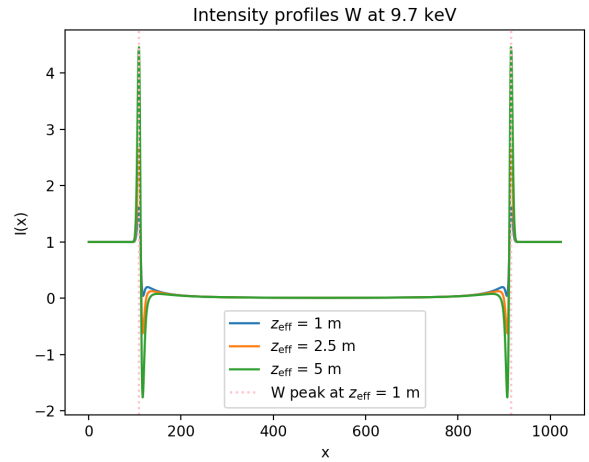


The fringes are probably too tall to represent real physics,
but I am not super sure.

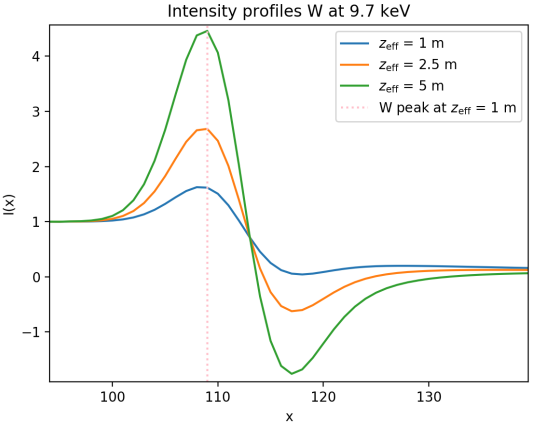
In the 2 *enhanced* figures below:
I can observe that for z_eff = 2.5m and z_eff = 5m the apparent brightest fringes at this energy have NOT moved outward as expected.

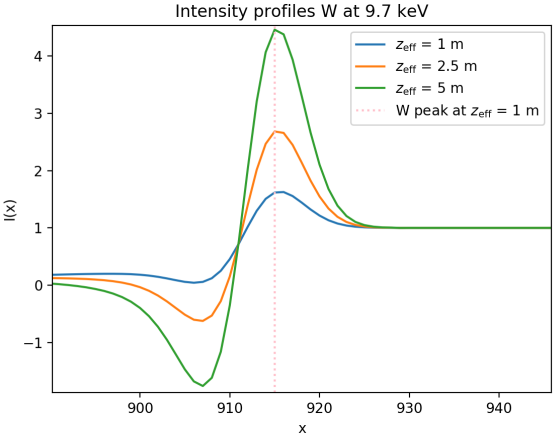


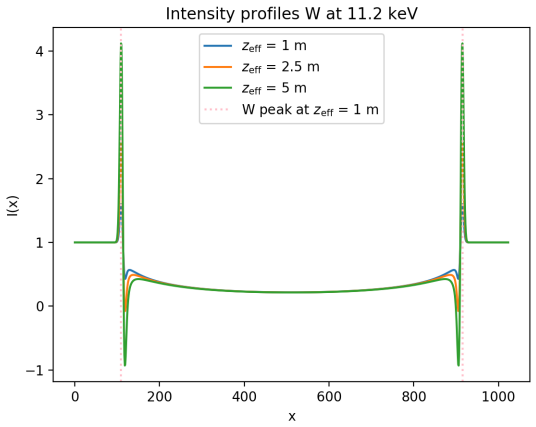




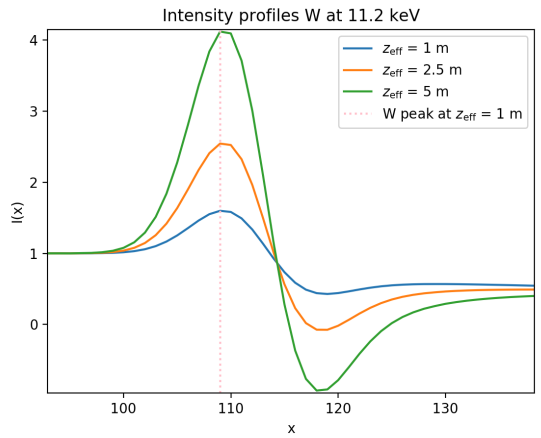
I think that the xri TIE is looking pretty

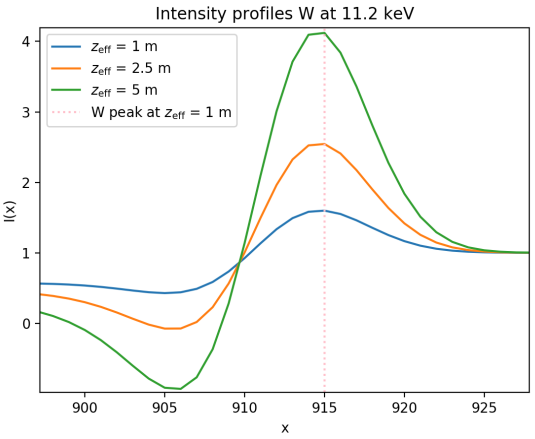




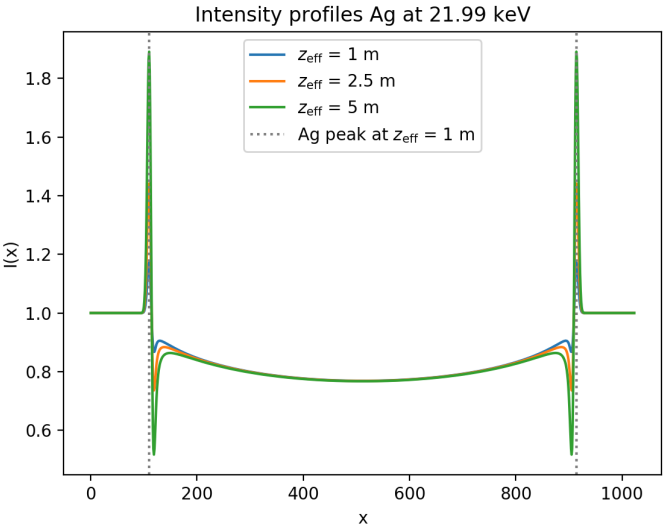


ALL GOOD...

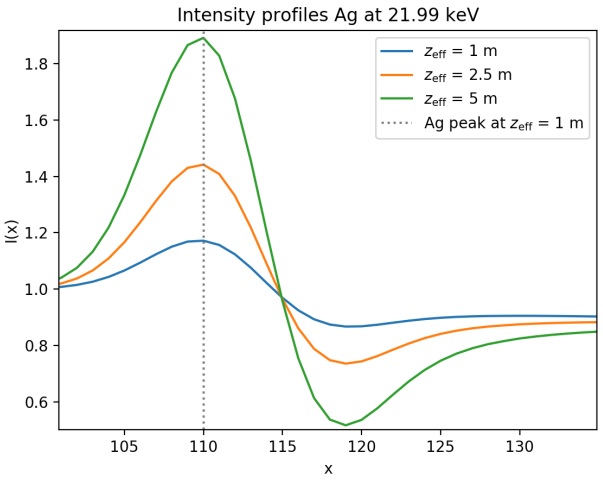


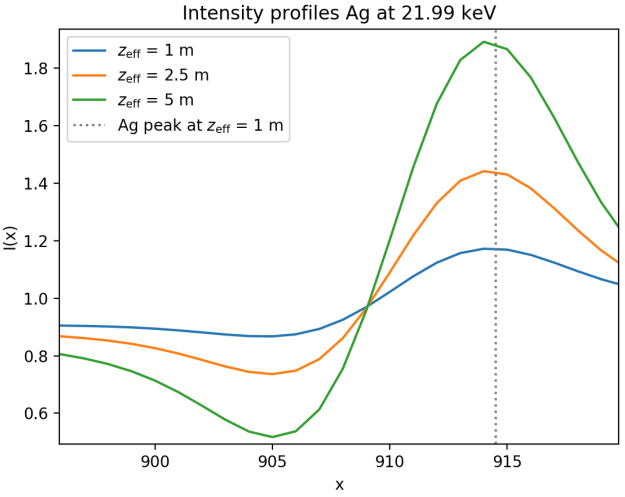


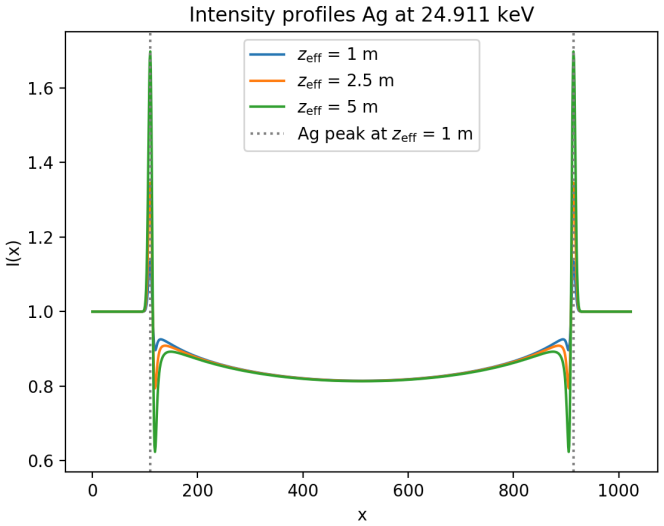
SILVER (Ag)



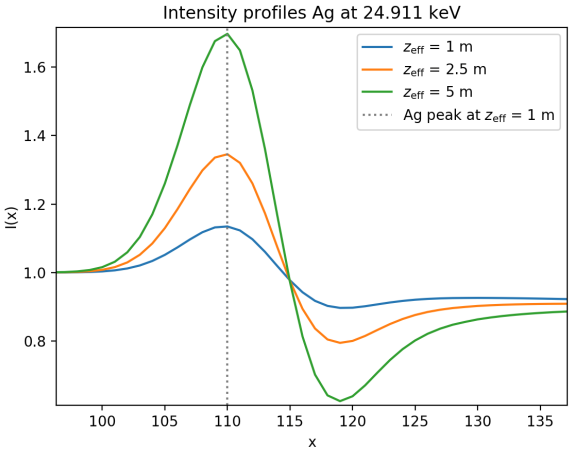
yup, all good again.

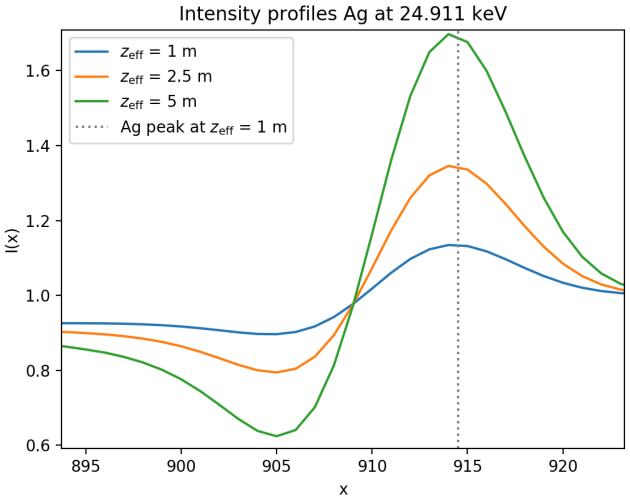






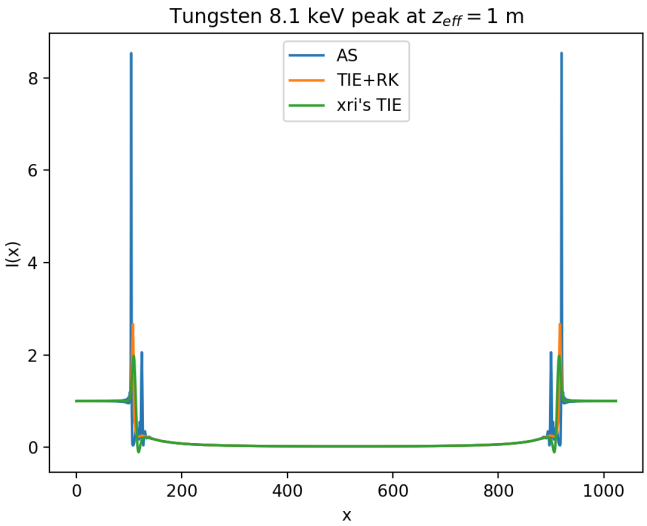
nothing weird is happening here...





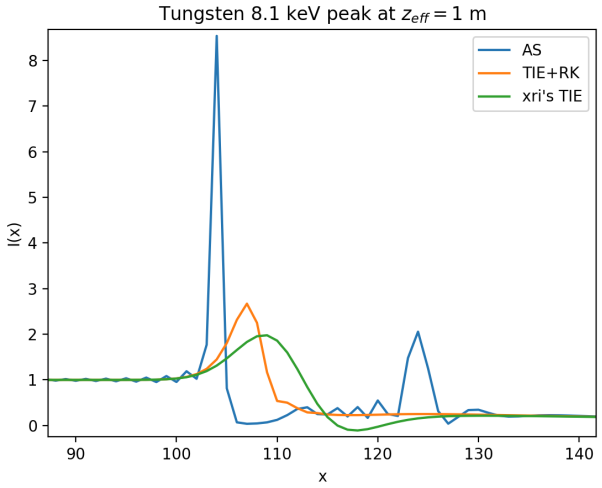
Comparing the three methods together

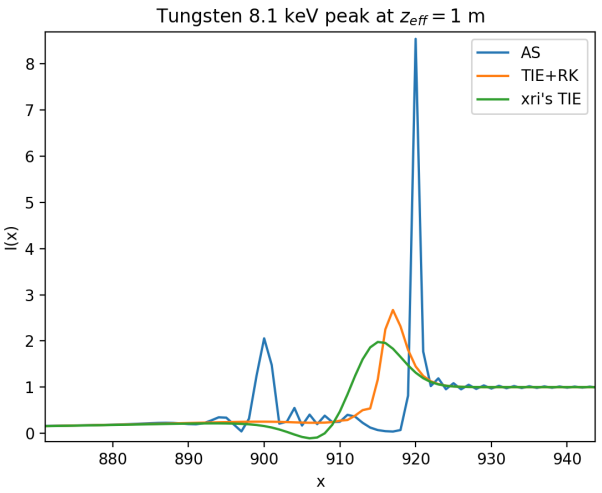
Comparing AS vs TIE+RK vs xri's TIE

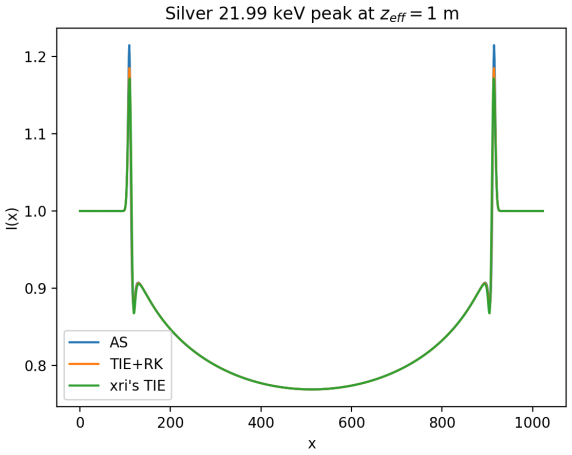


Here I tested xri's **AS**, **TIE+RK** and xri's **TIE** with
T = gaussian_filter(T, sigma=**4**)
NO ZOOM resampling

1. **TIE+RK** and xri's **TIE** appear smooth
2. **TIE+RK** has brighter fringes than xri's **TIE**
3. **AS** appears to have unphysically high fringes
4. **AS** has more ringing than any of the other methods

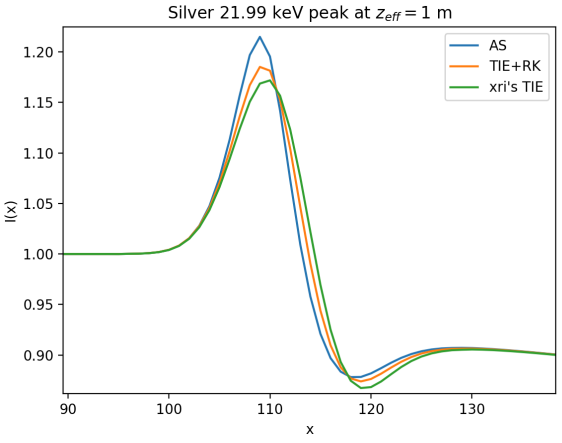


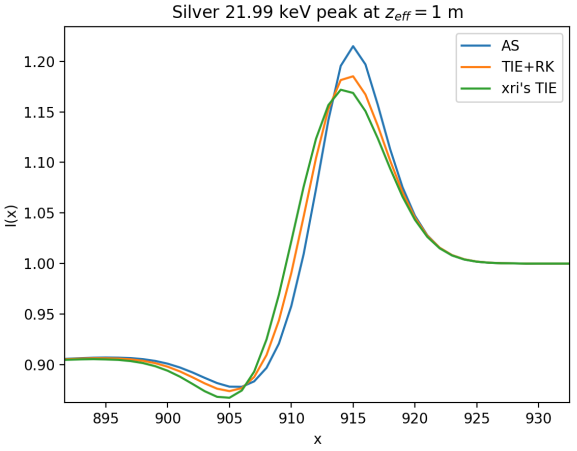




Here I tested xri's AS, **TIE+RK** and xri's **TIE** with
T = gaussian_filter(T, sigma=**4**)
NO ZOOM resampling

1. **ALL methods** appear smooth
2. **TIE+RK** has brighter fringes than xri's **TIE**
3. **AS** appears to have the brightest fringes





Mid-Sem Week (27/09/21 –03/10/21)

Aims:

1. Create simulations of phase contrast imaging for different targets

Tasks:

1. Meetings on **Monday 27/09** and **Wednesday 29/09**
   a. ~~Group: **CANCELLED**~~
   b. ~~one-on-one: **11 am**~~
2. **Read PyQt GUI book**
3. ~~Try W 8.1 for 5m with a smaller delta_z~~
4. ~~TEST different z_final AS vs TIE vs TIE+RK~~
5. ~~Try downsampling (zoom) again!~~
6. *Are we likely to see fringes shift in MK's lab?*

MK's latest comments

Wow - lots to digest here! In summary, your results are extremely impressive! It's really amazing to see your solution is clearly superior to our usual TIE algorithm as it is much closer to the AS than our TIE, particularly by allowing the fringes to shift position as they should. From these simulations, it looks like your TIE+RK is also more robust against ringing artefacts. Together, that's brilliant and very impressive - well done :)

Regarding the ringing, however, the downsampling (zooming) should help remove a lot of the ringing you're seeing, so we should look at that again. I'm not sure why this would cause the asymmetry you mentioned, but I'd like to see this when we meet.

Looking at the last panel, it looks like AS is likely still the best solution under ideal conditions. This is not surprising since it is a discrete version of the exact solution to the Helmholtz wave equation. But robustness is important too, so perhaps the TIE+RK is the most reliable solution? We need to look at the Zooming again to explore this further. Also, ***one other odd thing is the Tungsten 8.1 keV data where the TIE+RK has a nice peak, but no trough compared to the TIE. Then again, the TIE trough also looks like it goes slightly negative! It's hard to tell which is more reliable there without seeing actual numbers.***

In summary, I'm really impressed to see that the TIE is a better solution than I had appreciated, but only when you add in the RK to take the derivative to sufficiently high order terms. This is a great finding you have made, and is most impressive for a third year student - especially since using RK was your idea, not mine!

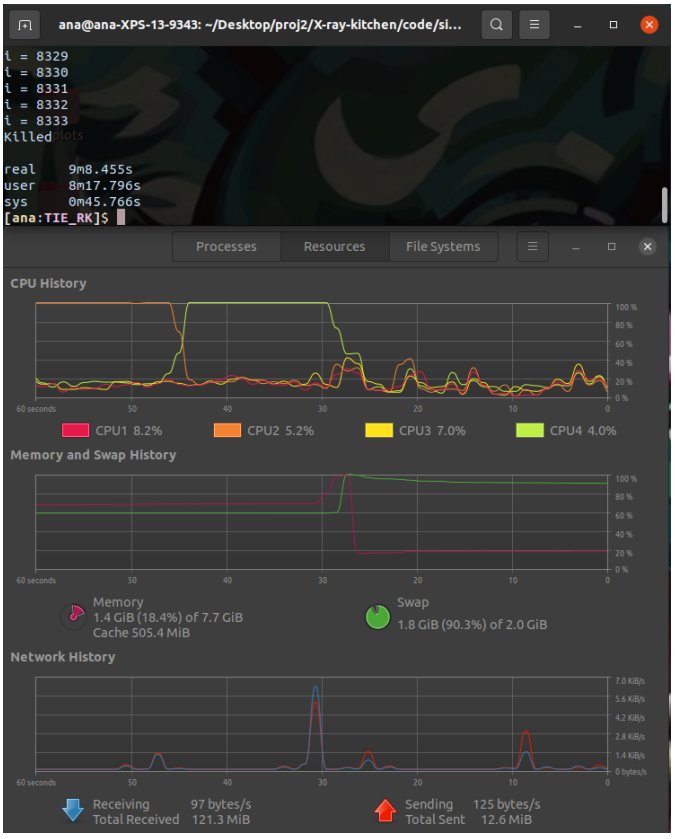
The main question to consider is ***how this is useful****. It definitely appears better than our TIE for doing simulations, and it may be more robust than the AS. Perhaps AS is only as robust if you downsample, and then the point may be a question of speed vs accuracy. For some applications, speed can be more important than accuracy, but I've not really come across that situation for these types of simulations.* ***However, no one has successfully developed a phase retrieval algorithm based on the AS, but there are many such solutions using the TIE. One example is attached where the authors did use higher order terms instead of the finite difference approximation to improve phase retrieval, but it required multiple images to be acquired.*** *This is a problem for X-ray imaging as it will increase the radiation dose. David Paganin's single image algorithm (attached) is ideal as it is noise robust and only requires a single exposure, but is based on the TIE with the finite difference approximation. I don't know that the RK approach can help with phase retrieval, but I'd certainly like to think about that more. Feel free to do so, but not at the cost of getting your project done on time!*

So where to from here? Remind me, ***did your simulations show yet if we are likely to see fringes shift in my lab? I'm not sure if you finished those yet. The idea was to look at tungsten at, say, 35 kV and see if the low energy spectral peaks would create phase contrast peaks that have a different pixel position for phase contrast peaks than the mean energy from the Bremsstrahlung radiation peak at around 21 keV. If they do, we can use the photon counter to see if we can observe this effect experimentally. <---???? WHAT***

Smaller evolution steps TIE+RK

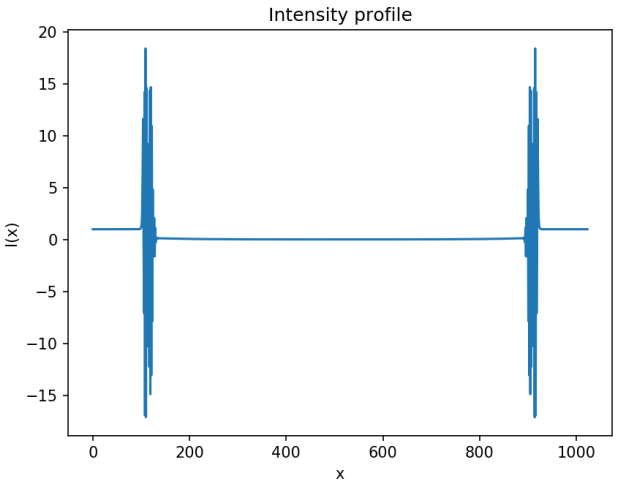
I want to verify my suspicion on the current result for the highly unstable case: Tungsten 8.1 keV at 5 m

```
z_final = 5 * m # propagation distance
delta_z = 0.6 * mm  # (n_z = ~8333)
```



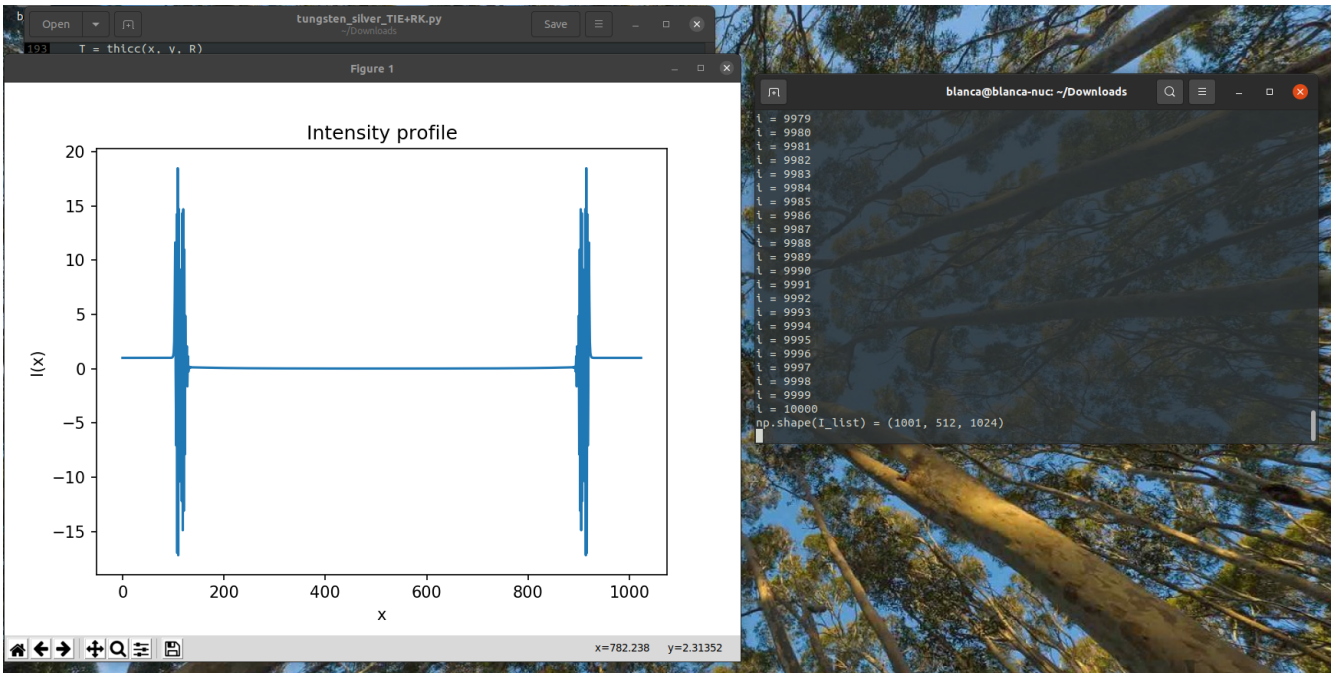
←Process is killed

NOTE : The big spike in memory usage... therefore, the process brings my laptop to the limit. The funny thing is that my laptop can calculate this many steps but cannot take a screenshot at the same time as bringing the plot up after it finishes propagating over 833 steps.
Anyway, the output is not obviously better than before.



Using my mum's computer I set delta_z = 0.5 mm and there was no visible improvement

```
z_final = 5 * m # propagation distance
delta_z = 0.5 * mm  # (n_z = 10000)
```

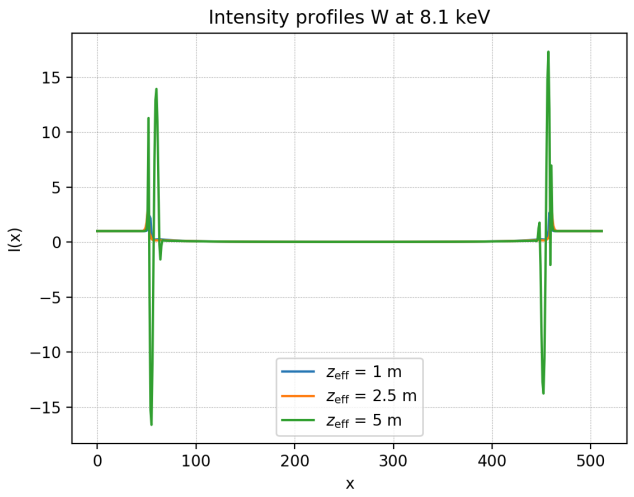
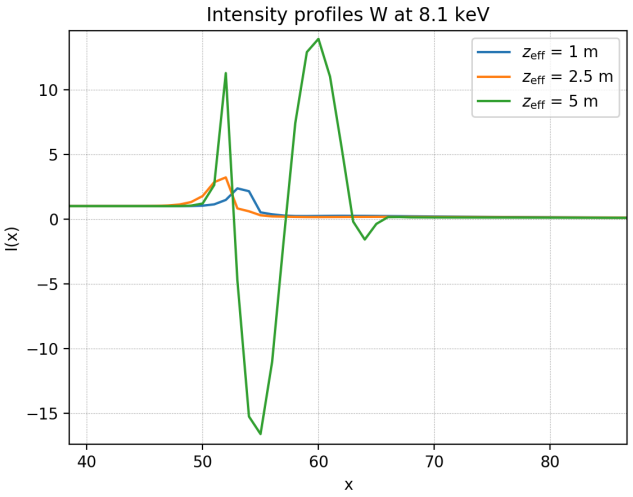
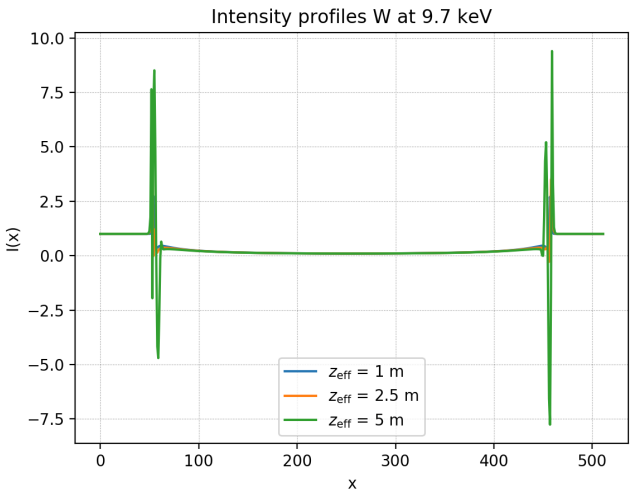
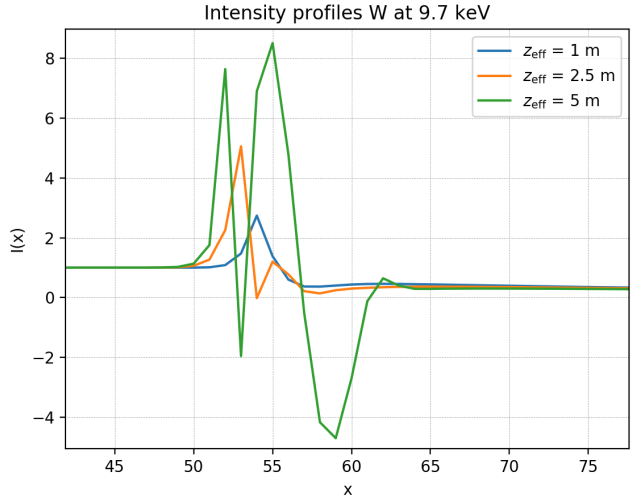


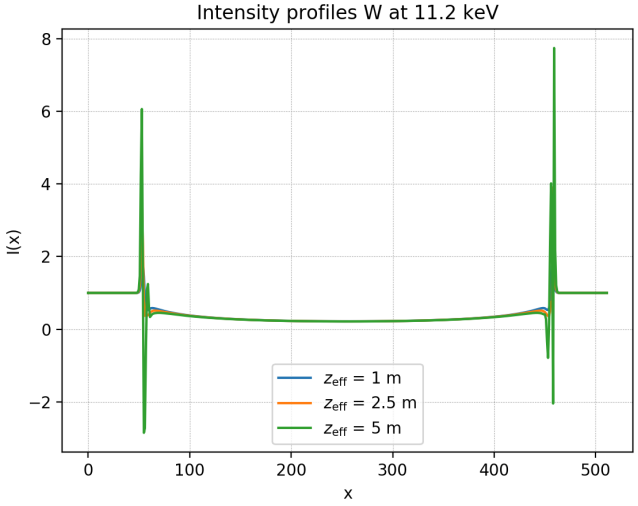
Maybe I could decrease delta_z by a larger factor but this might not be worth the computational time.

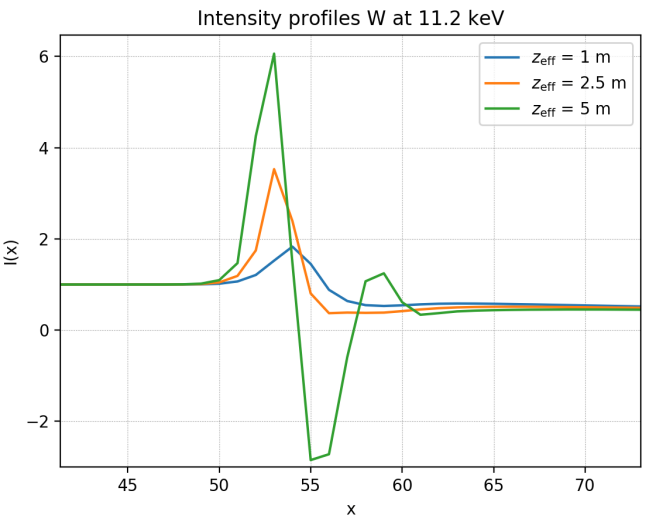
Try downsampling again!

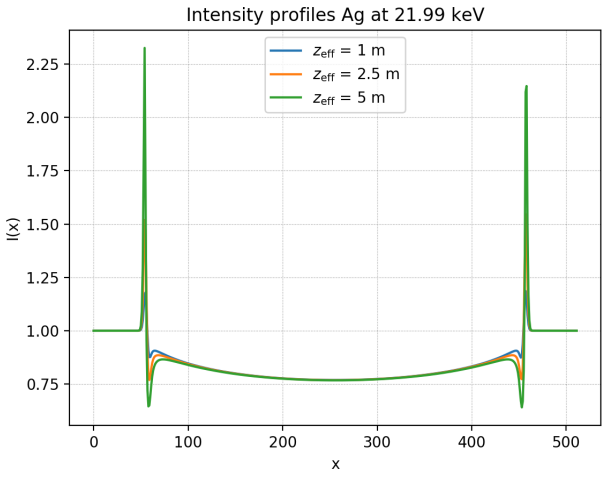
zoom(*Intensity*, 0.5, order=3)

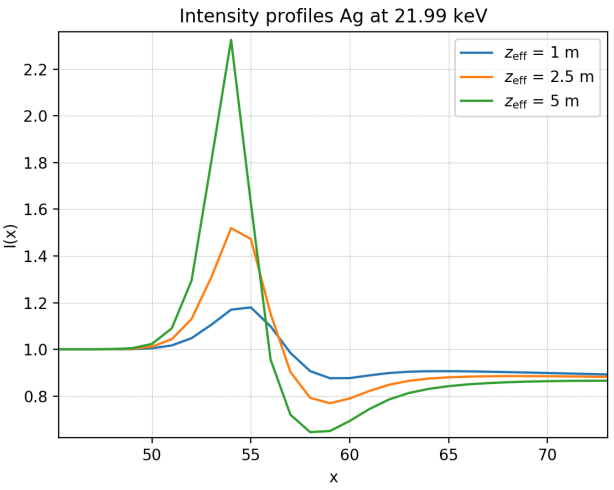
Notice the asymmetry

Full view	Left peaks *enhance!*
	
	



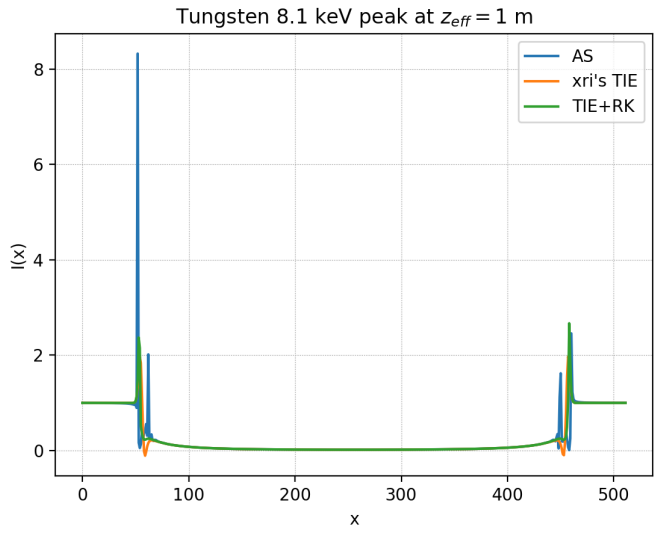


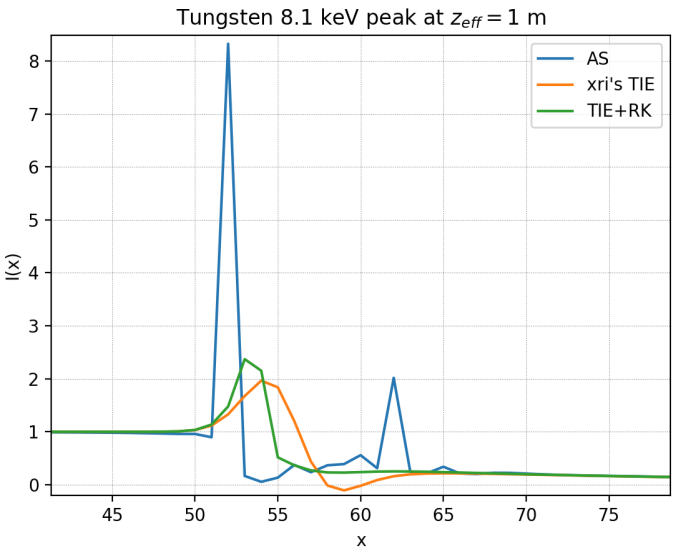


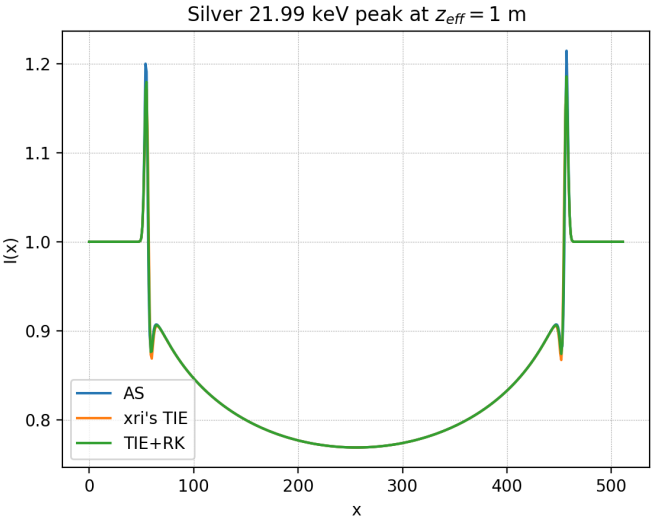


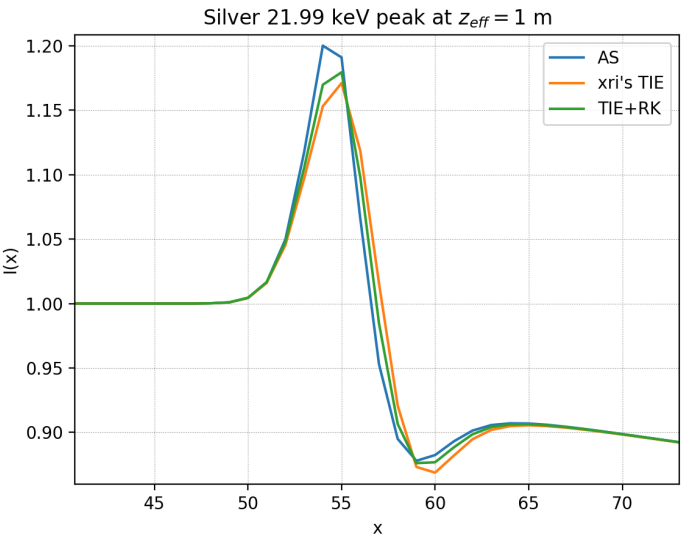
TEST: ALL methods at 1m

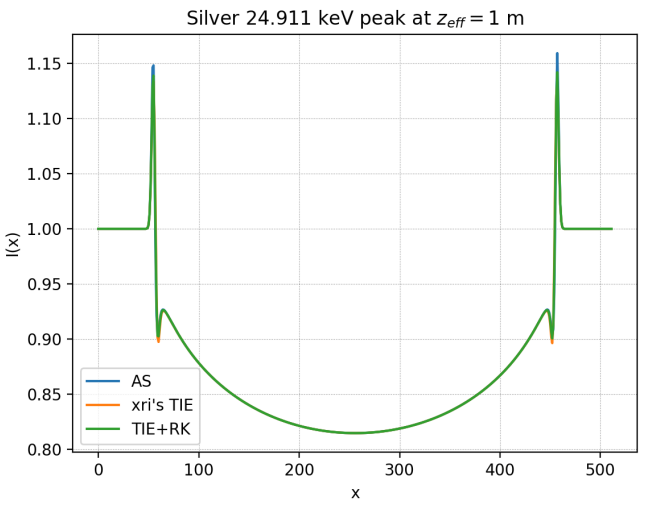
Full view	Left peaks *enhance!*

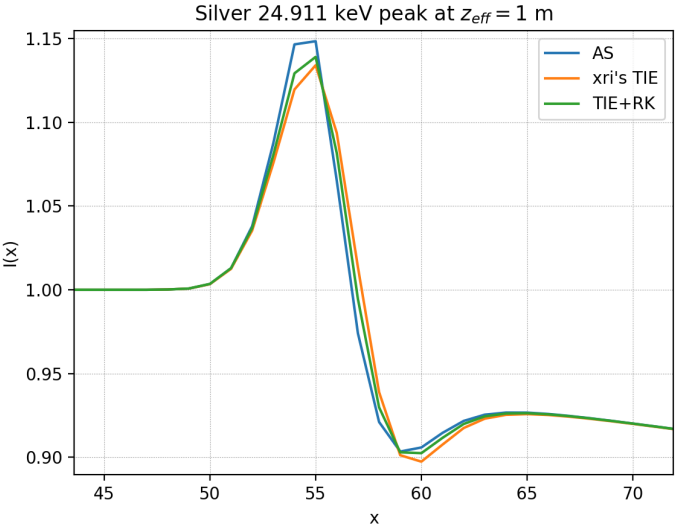






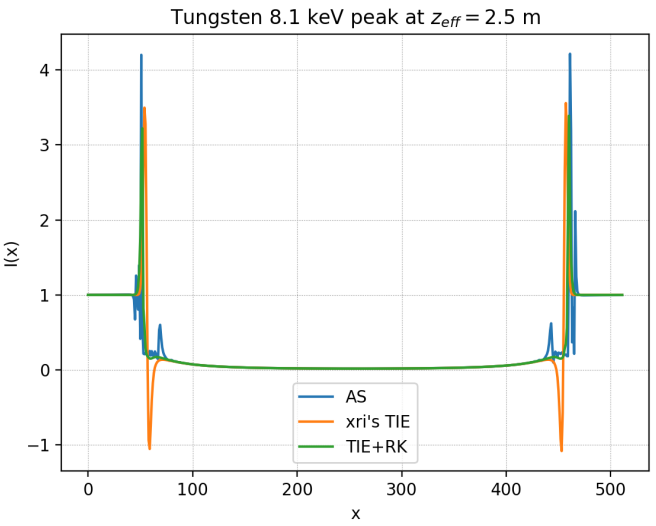




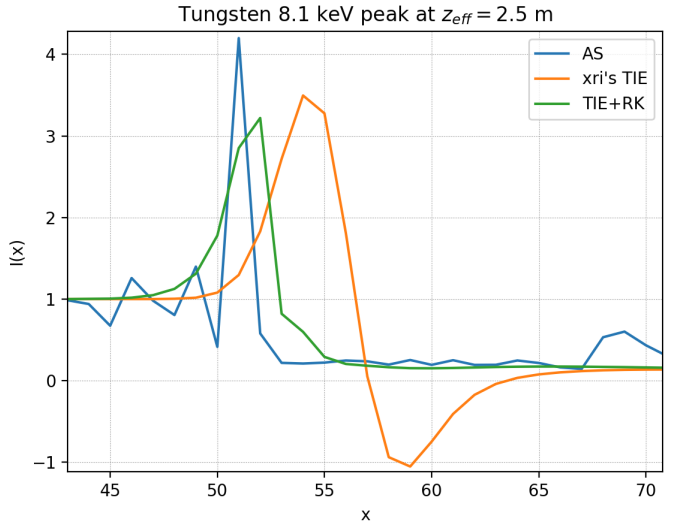


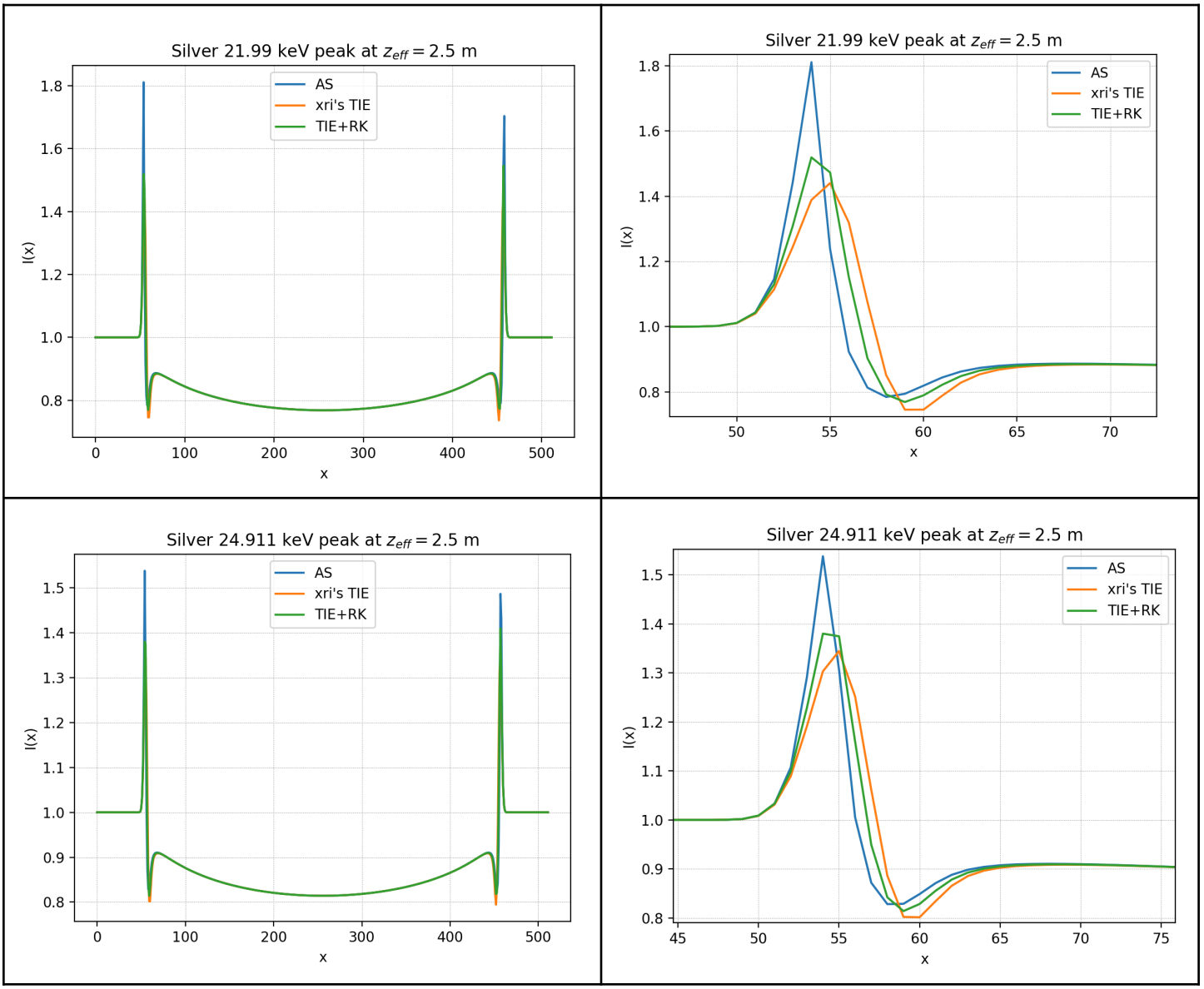
TEST: ALL methods at 2.5m

Full view



Left peaks *enhance!*





29/09/2021

MEETING DAY!

Things to address today:

Regarding the ringing, however, the downsampling (zooming) should help remove a lot of the ringing you're seeing, so we should look at that again. I'm not sure why this would cause the asymmetry you mentioned, but I'd like to see this when we meet.

ALIASING?

Maybe shift the data forward and backward a few pixels and see if asymmetry and other undesirable visual effects are decreased.

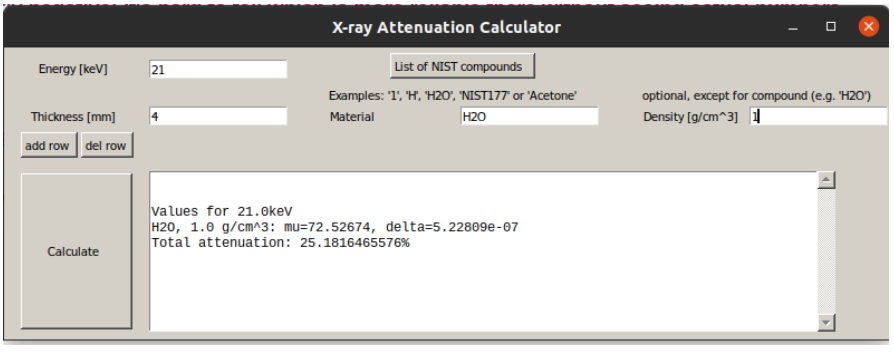
Or maybe take the average of n pixels another way?

Odd thing: The tungsten 8.1 keV data where the TIE+RK has a nice peak, but no trough compared to the TIE. Then again, the TIE trough also looks like it goes slightly negative! It's hard to tell which is more reliable there without seeing actual numbers.

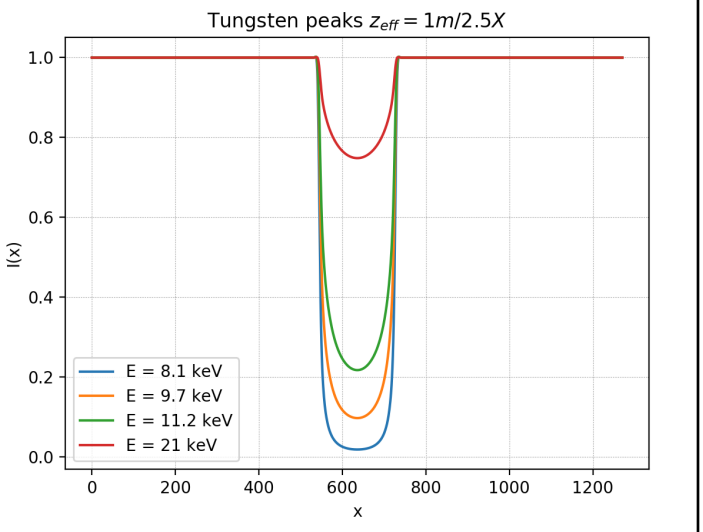
TODO:

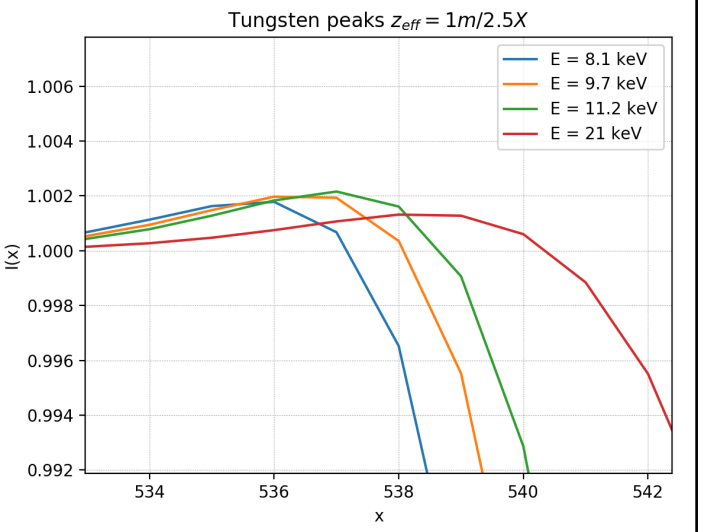
1. ~~Bremsstrahlung radiation peak - W + (8.1, 9.7, 11.2) -> averaging phase contrast fringes~~
2. ~~Can I make TIE+RK faster??~~
3. Can TIE+RK be used for phase retrieval?
    a. READ ABOUT PAGS ALGORITHM
        i. Maybe we can implement RK propagation there

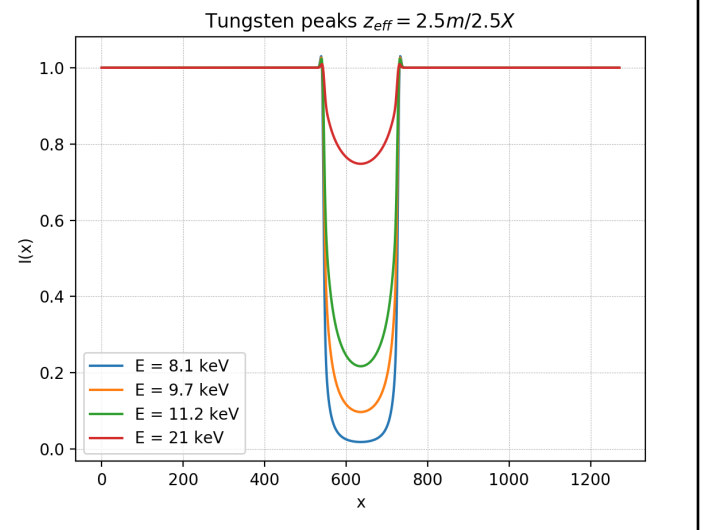
Tungsten Bremsstrahlung peak + characteristic peaks averaging

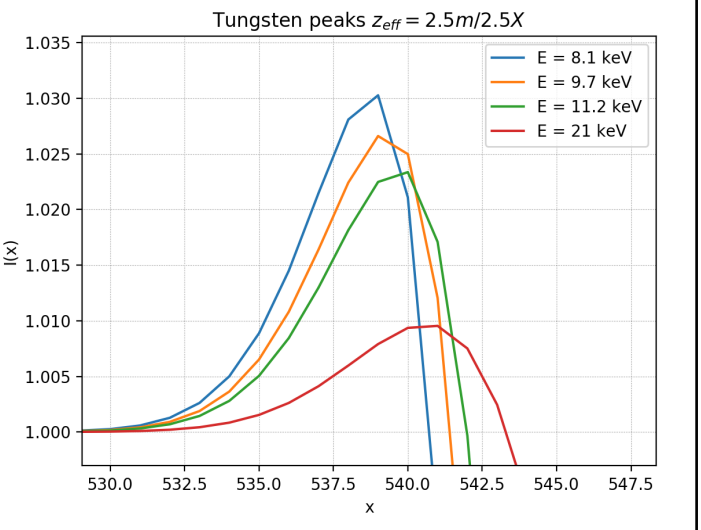


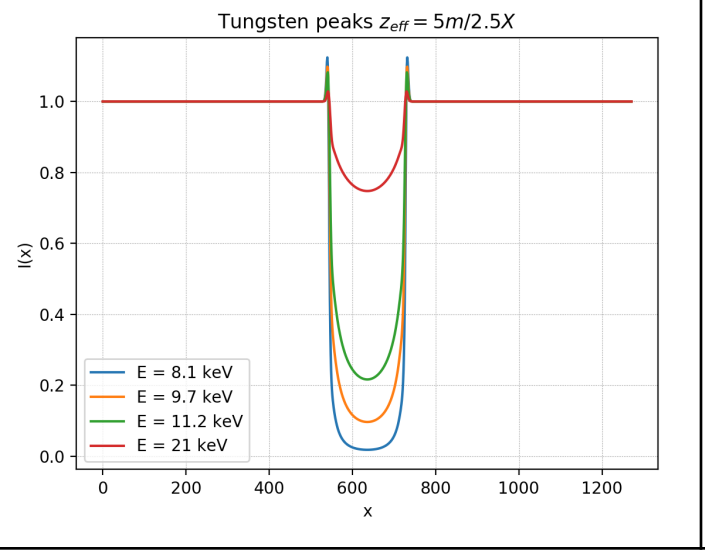
 # # Matching LAB parameters
Magnification
M = 2.5
# x-array parameters
delta_x = 55 * um / M
x_max = 35 * mm / M
x_min = -x_max
n_x = int((x_max - x_min) / delta_x)
x = np.linspace(-x_max, x_max, n_x, endpoint=False)
# # y-array parameters
delta_y = 55 * um / M
y_max = 7 * mm / M
y_min = -y_max
n_y = int((y_max - y_min) / delta_y)
y = np.linspace(-y_max, y_max, n_y, endpoint=False).reshape(n_y, 1)

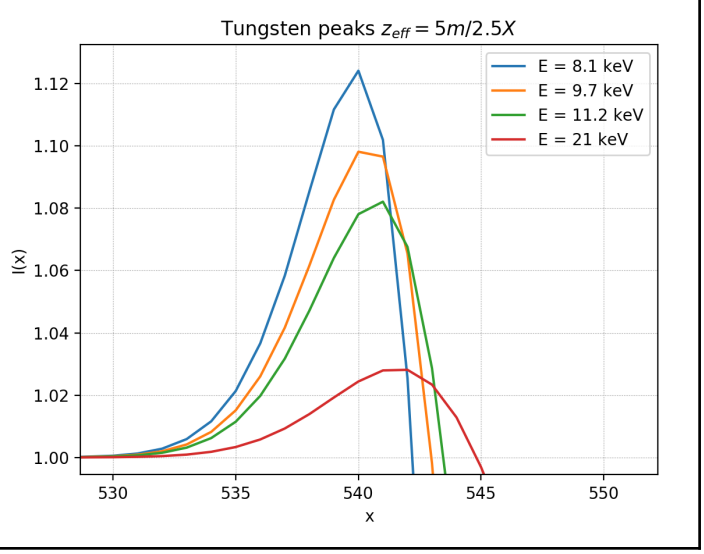








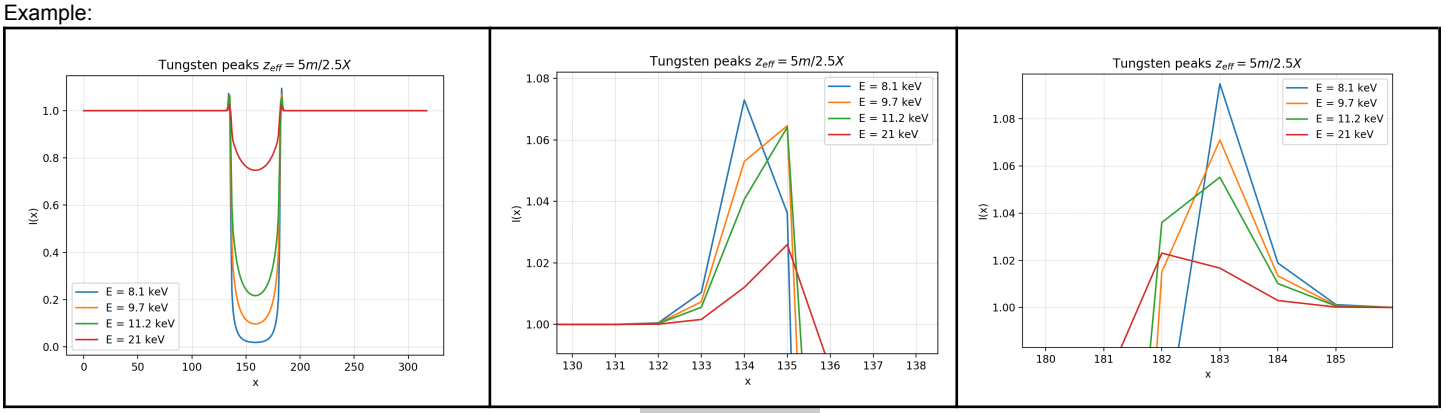




To avoid the visible aliasing asymmetry due to downsampling, I wrote this:

```
array = array.reshape(int(len(array ) / 4), 4).mean(axis = -1)
```

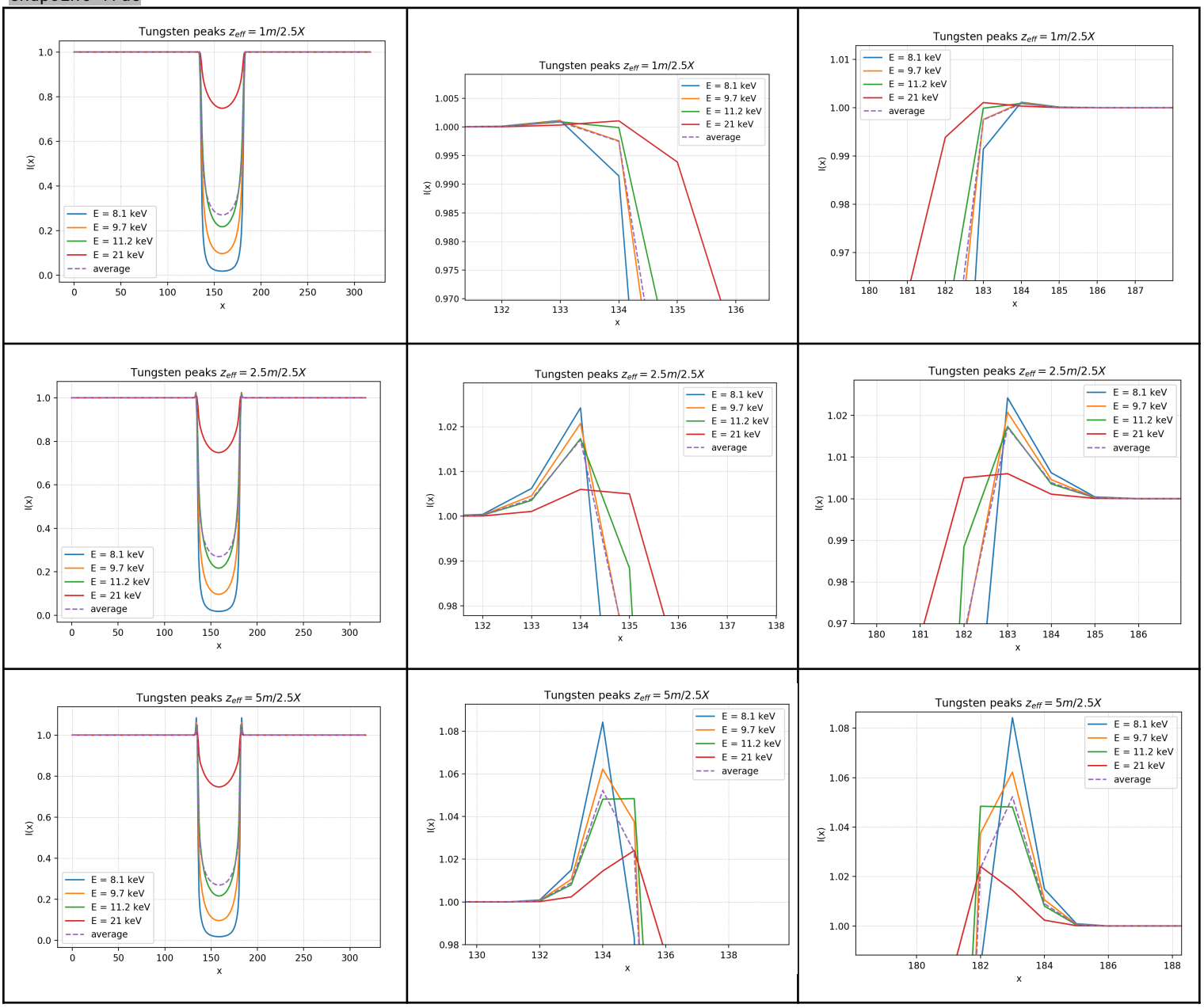
Here I take the 1D array data (crosssection intensity function of x) and reshape it into a (n_array / 4 , 4) array
After this I take the mean value of the 4 values in each row. This **minimises** the aliasing occuring by **downsampling**.
Example:



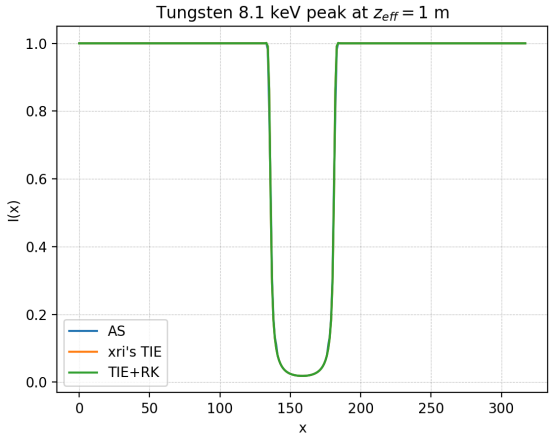
I can still see some asymmetry :(I reckon that I have to stop using `endpoint=False` in my x and y arrays.

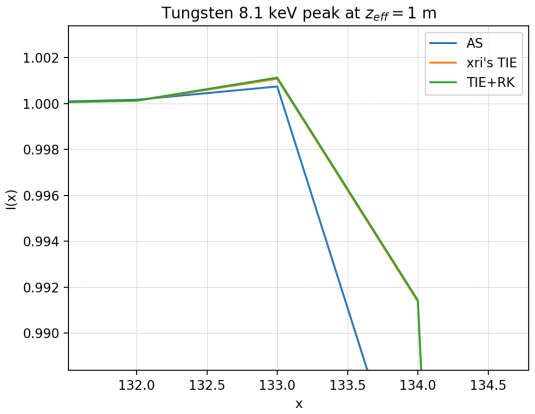
Plotting all peaks and the average

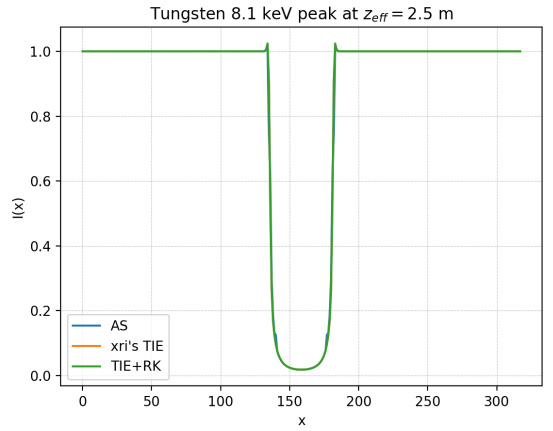
`endpoint=True`

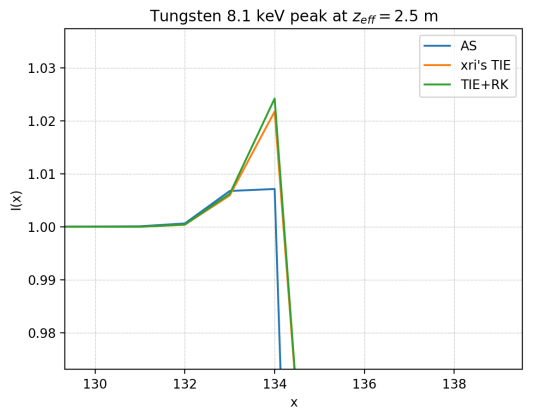


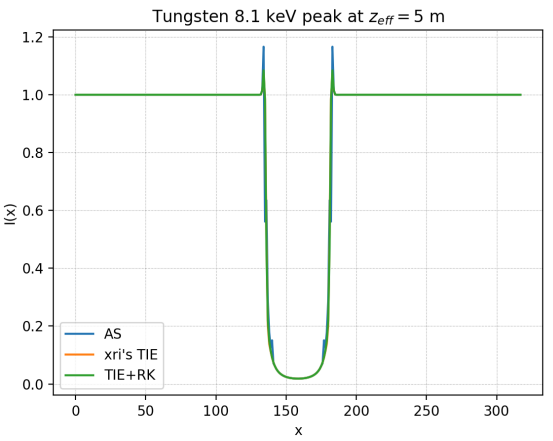
Yay symmetry!

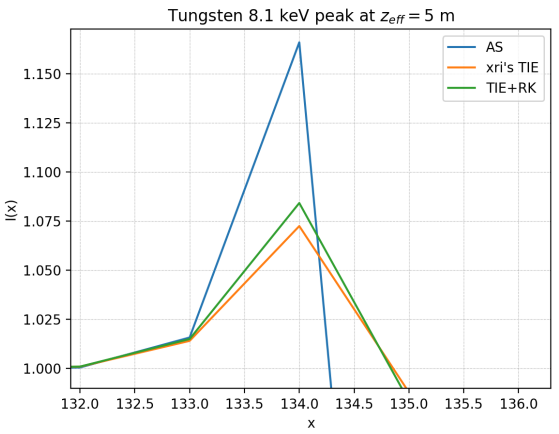


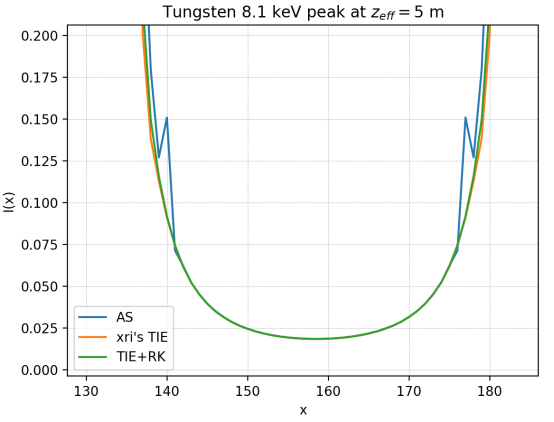












TIE+RK

Consistently gives the most stable and highest contrast solution for this energy.

01/10/2021

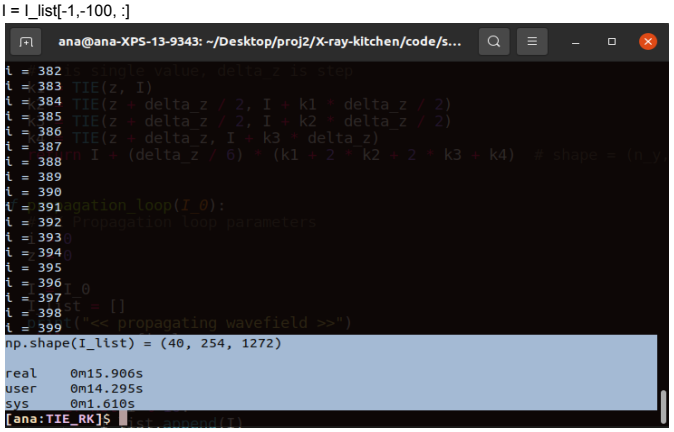
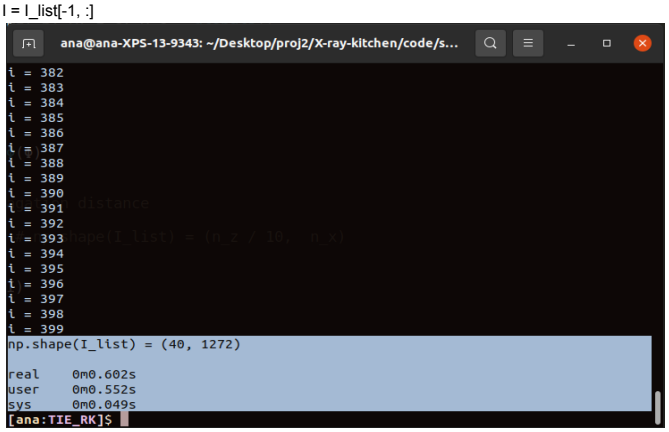
Can I make TIE+RK faster?

I can probably decrease the time spent in propagation by

1. ~~**Eliminating the dependence in y**~~
    a. I always end up plotting only the phase contrast along x
    b. This might decrease the time by not evaluating `np.gradient` in y
2. Only evaluating the cylinder and not also the surrounding region?

TIE+RK (x,y)	TIE+RK (x)

M = 2.5
z_final = 1 * m / M # eff propagation distance
delta_z = 1 * mm
I_list = propagation_loop(I_0) # np.shape(I_list) = (n_z / 10, n_y, n_x)
 	
I = I_list[-1,-100, :]
	M = 2.5
	z_final = 1 * m / M # eff propagation distance
	delta_z = 1 * mm
	I_list = propagation_loop(I_0) # np.shape(I_list) = (n_z / 10, n_x)
	
	I = I_list[-1, :]

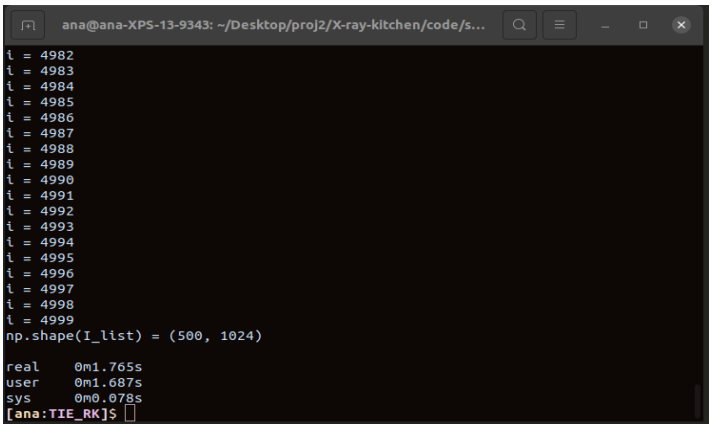
Comparing time performances

# Material = water, density = 1 g/cm3** 	
E = 8.1 # keV ---> W @ 35 kV
δ = 3.52955e-06
μ = 999.13349 # per m
	# Theoretical discretisation parameters
	# # x-array parameters
	n = 1024
	x_max = (n / 2) * 5 * um
	x = np.linspace(-x_max, x_max, n, endpoint=True)
	delta_x = x[1] - x[0]
	# # y-array parameters
	y_max = (n / 2) * 5 * um
	y = np.linspace(-y_max, y_max, n, endpoint=True)
	delta_y = y[1] - y[0]
	y = y.reshape(n, 1)

xri's TIE (2D)

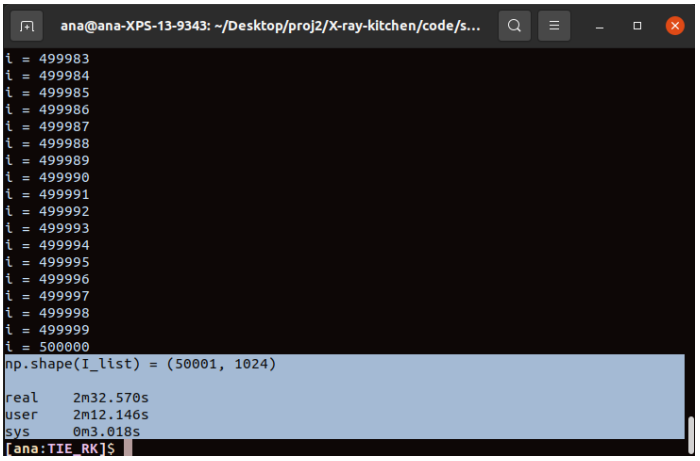
AS (2D)

TIE+RK (1D)

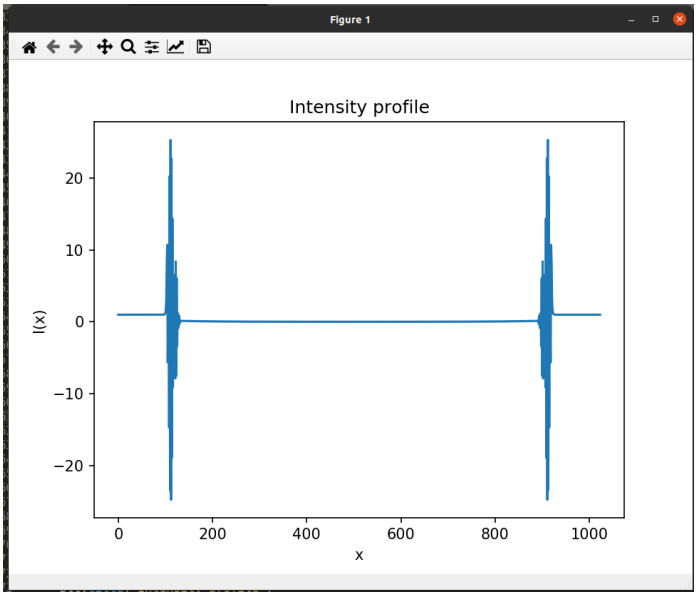


IT LOOKS LIKE MY METHOD IS THE FASTEST OF THE THREE WITH THIS CHANGE.

I can now afford to check if decreasing the z step size improves the resolution of W: 8.1 keV for 5m propagation



NOPE… there is no improvement if I increase delta_z by itself.
I probably need to correct the x direction size too...

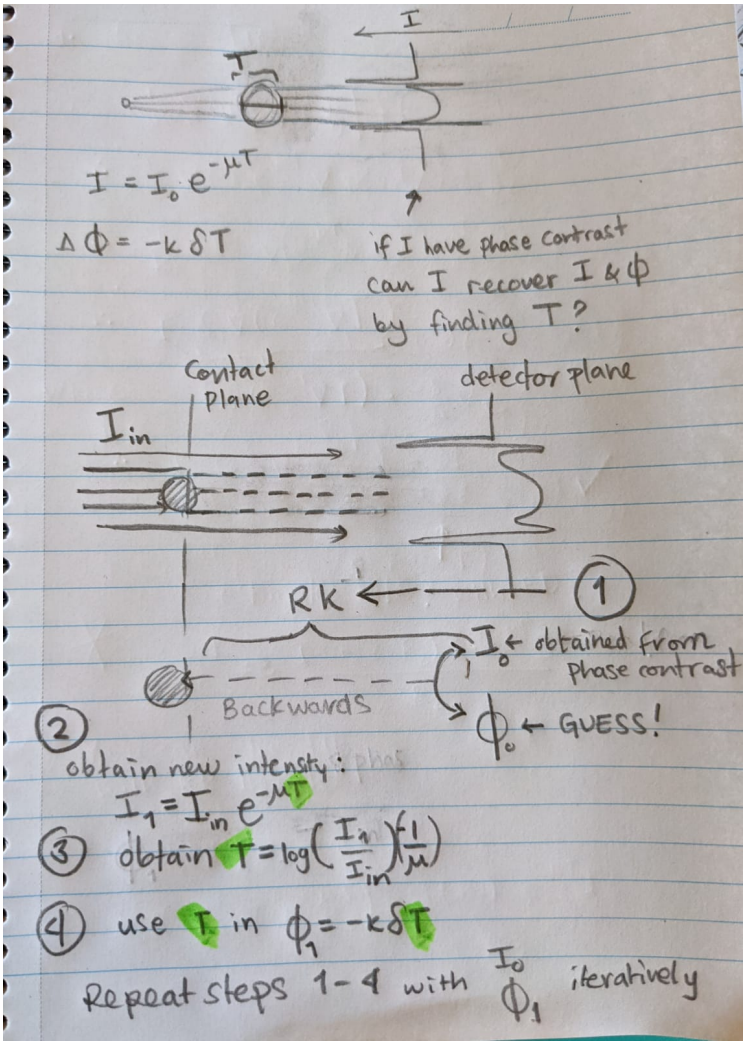


Using TIE+RK for phase retrieval

02/10/2021

While reading “Simultaneous phase and amplitude extraction from a single defocused image of a homogeneous object” by D Paganin et al., I noticed how clever the phase retrieval method really is! The algorithm links the BBL and the phase shift equation from the projection approximation by the thickness of the imaged sample. Using the TIE and approximating the intensity longitudinal derivative in an Euler step, the algorithm somewhat combines the AS and the TIE into a single expression that recovers the Thickness of the imaged sample.

I don’t think that I can come up with something that elegant at this time. What I am going to do instead is a sort of **brute force iterative** approach to recovering the phase.



Here, like Pags I make use of the TIE and the projection approximation expressions for the phase and intensity.

I recover the object’s thickness (T) using the phase contrast intensity by back propagating with RK.

I use the resultant T from the “recovered” intensity to calculate a new phase which will theoretically be closer to the original incident phase coming from the source.

My algorithm is quite simple. I hope it is not ridiculously slow.

03/10/2021

I wrote this algorithm as a 1D process

These are the globals:

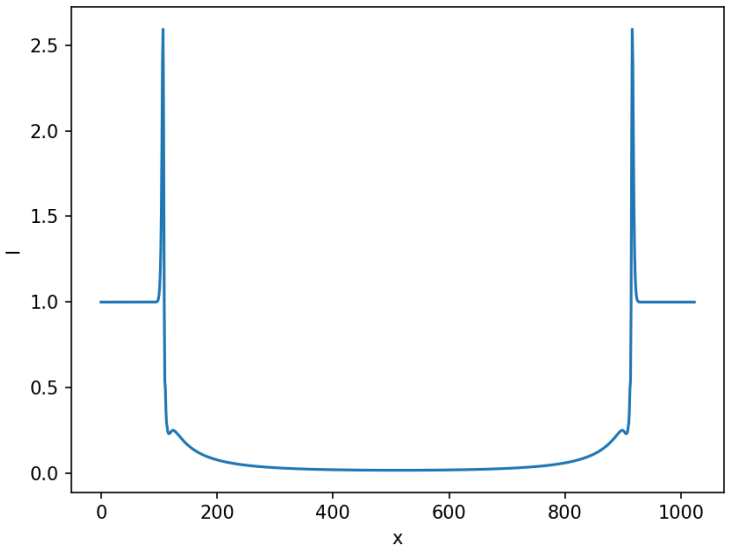
```python
if __name__ == '__main__':

    M, x, n_x, delta_x, δ, μ, k, I_in = globals()

    # # Step 1: from phase contrast obtain I_0
    # I_0 = np.load("5m_I1_1_M=2.5.npy")
    # I_0 = np.load("5m_I2_1_M=2.5.npy")
    # I_0 = np.load("5m_I3_1_M=2.5.npy")
    I_0 = np.load("test_W.npy")


    # # # Step 1.2: guess a phase
    Φ = np.zeros_like(x)
    dΦ_dx, lap_Φ = gradΦ_laplacianΦ(Φ)

    # # Step 2: run 4th order RK towards contact plane
    z_eff = 1 * m / M # eff propagation distance
    delta_z = 1 * mm


    Φ = phase_retrieval(I_0)
```

These are the 2 new functions I added:

```python
def phase_retrieval(I_0):

    global Φ, dΦ_dx, lap_Φ, first_iteration

    for i in range(3):
        first_iteration = i == 0
        print(f"{i = }")

        print(f"{np.shape(I_0) = }")
        I = back_propagation_loop(z_eff, delta_z, I_0)
        I = I[-1,:] # np.shape(I) = (n_x,)
        # print(f"{np.shape(I) = }")

        # # Step 3: obtain T from new intensity
        T = (-1 / μ) * np.log(I)# / I_in)

        # # Step 4: use T to calculate a new phase
        Φ = - k * δ * T
        print(f"{np.shape(Φ) = }")

        # new phase derivatives for TIE
        dΦ_dx, lap_Φ = gradΦ_laplacianΦ(Φ)

        # # Φ changes
        # plt.plot(Φ)
        # plt.xlabel("x")
        # plt.ylabel(R"$\phi(x)$")
        # plt.title("phase profile")
        # plt.show()

        # # I changes
        plt.plot(I)
        plt.xlabel("x")
        plt.ylabel("I")
        plt.show()
    return Φ
```

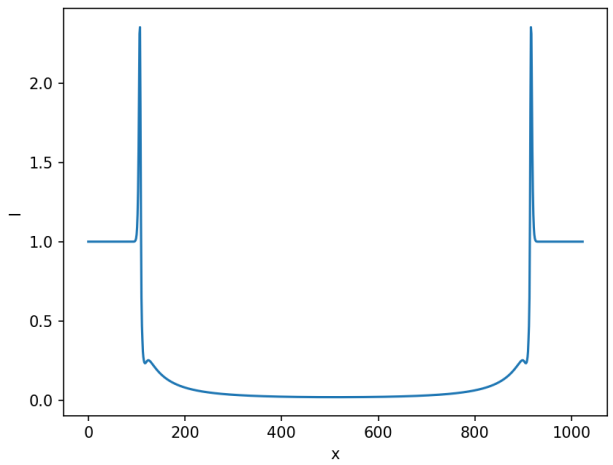
```python
def back_propagation_loop(z_eff, delta_z, I_0):
    # RK Propagation loop parameters
    j = 0
    z = z_eff

    # checking dimension of intensity array
    if len(np.shape(I_0)) == 1:
        print("<<< 1D array >>>")
        I = I_0
    else:
        print("<<< 2D array >>>")
        I = I_0[-1,:]

    I_list = []
    print("<< propagating wavefield >>")
    while z > 0:
        print(f"{j = }")
        # print(f"{z = }")

        # spatial evolution
        I = Runge_Kutta(z, -delta_z, I) # (n_x,)
        if not j % 10:
            I_list.append(I)

        if not first_iteration and not j % 10:
            plt.plot(I)
            plt.xlabel("x")
            plt.ylabel("I")
            plt.show()
        j += 1
        z -= delta_z

    I_list = np.array(I_list)
    return I_list # shape = (n_z / 10, n_x,)
```

Input phase contrast (I_0 = np.load("test_W.npy")**)**
i = 0, j = 0

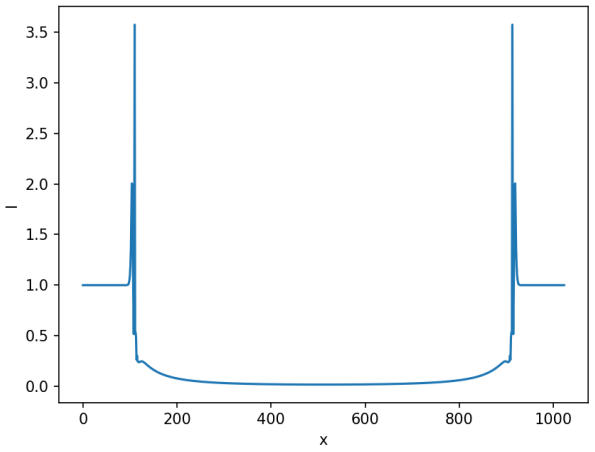


Output of second iteration (i = 1)

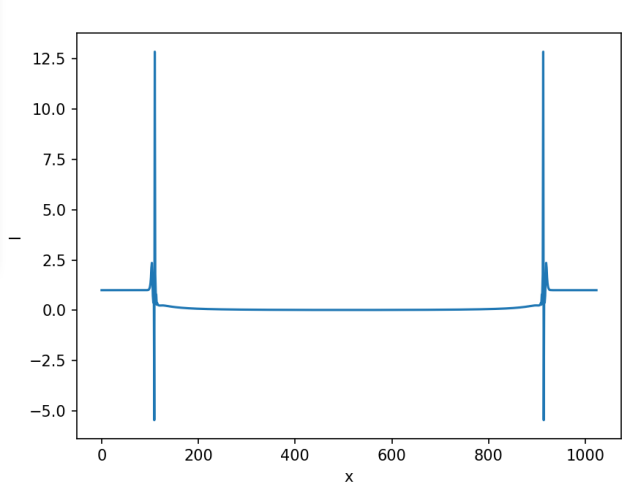
j = 0



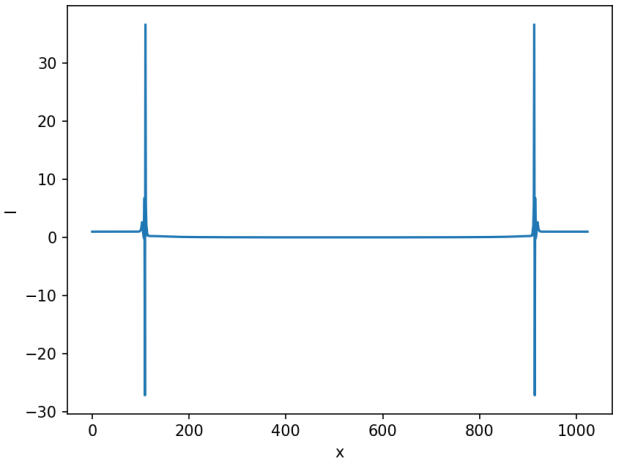
j = 10



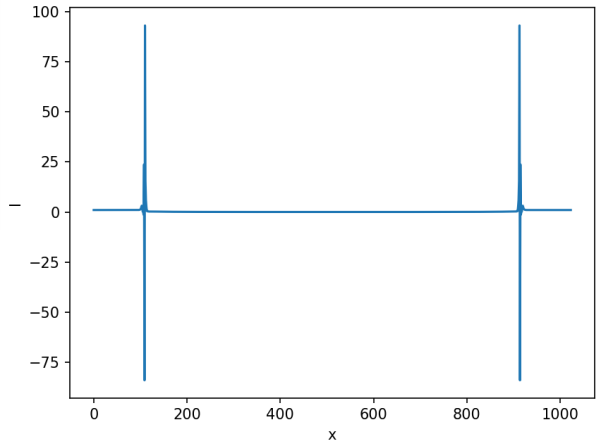
j = 20



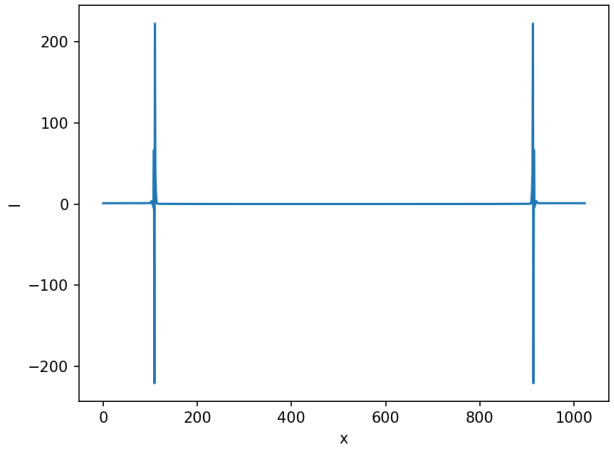
j = 30



j = 40

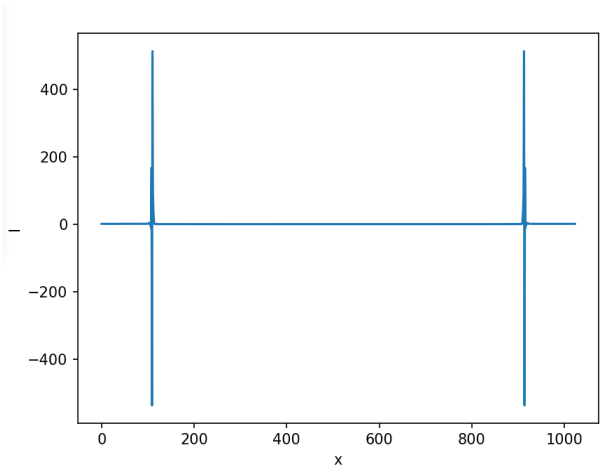


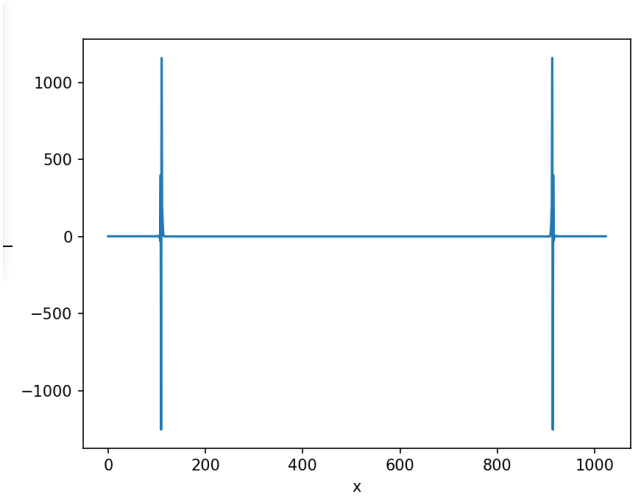
j = 50



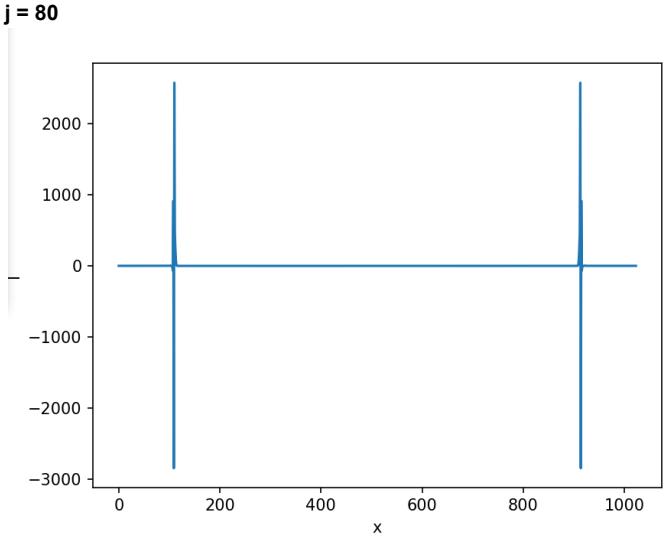
j = 60

j = 70

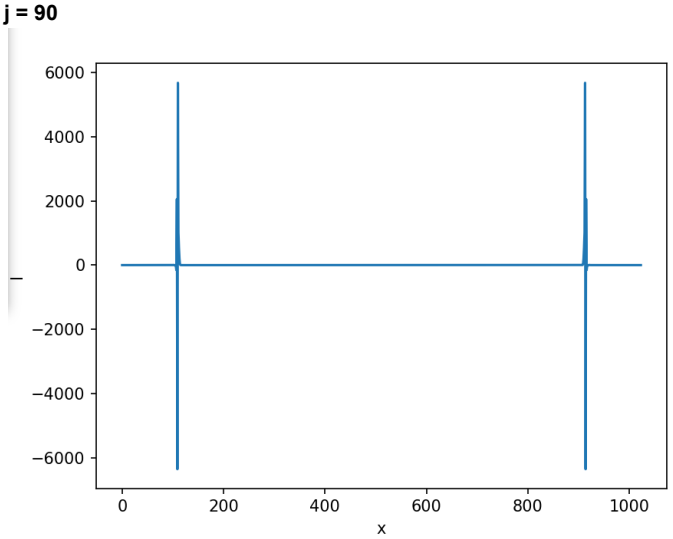




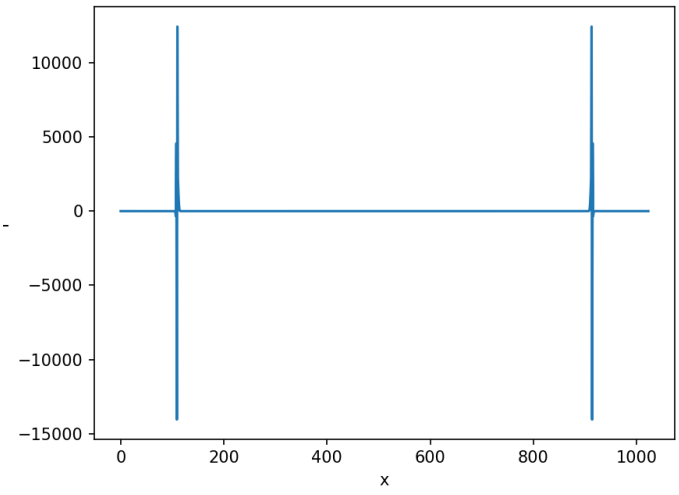
j = 80



j = 90



j = 100

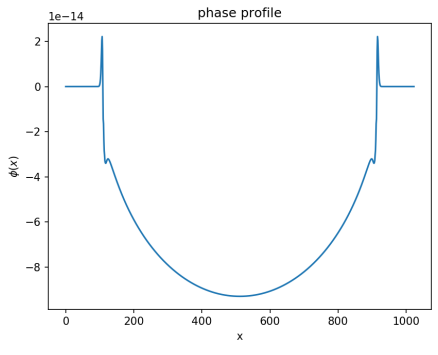


The intensity's magnitude explodes!?
This looks a bit confusing because I redefine I as I_0 every j step.

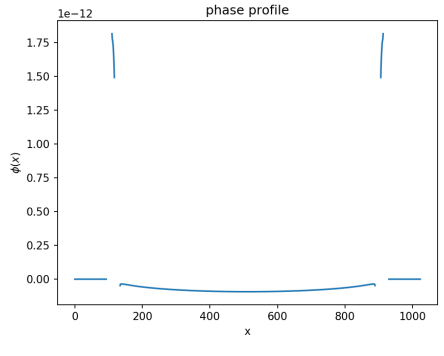
My algorithm looks very susceptible to numerical instability.

This is the phase after 3 iterations

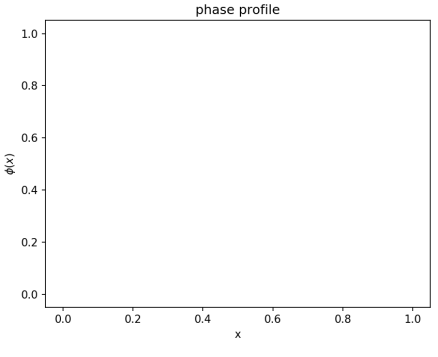
i = 0



i = 1



i = 2



I am sure I could think about this for a bit longer but I am going to start working on my report and presentation for the time being.

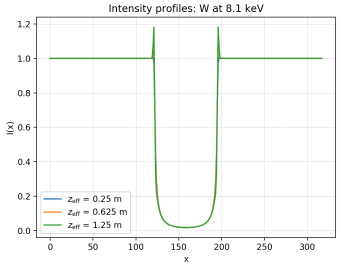
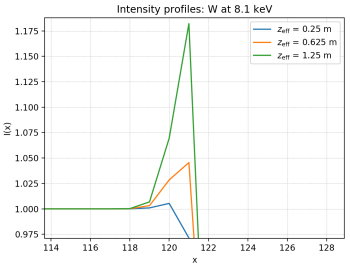
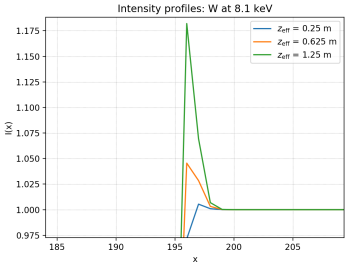
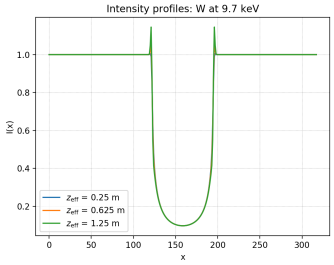
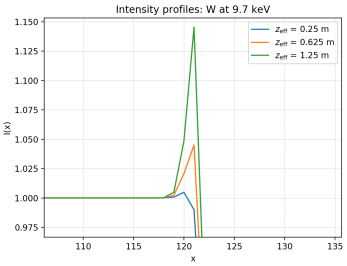
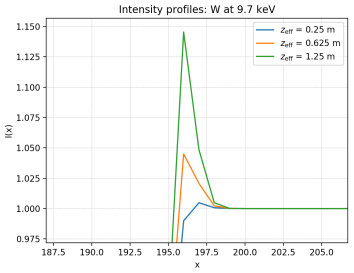
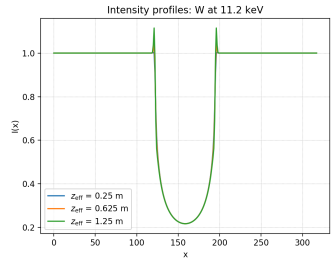
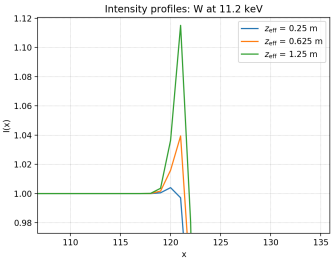
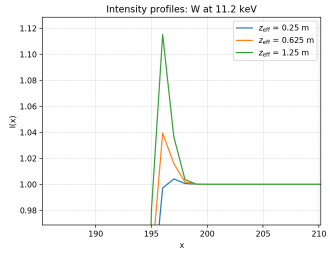
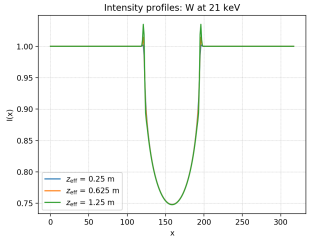
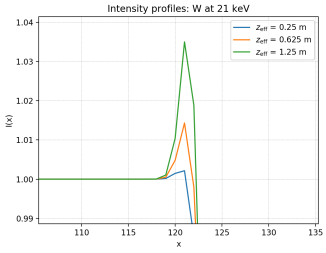
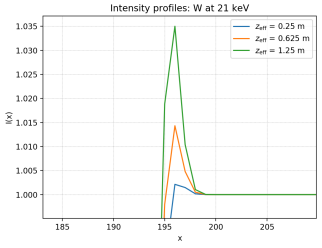
Week 10 (04/10/21 –10/10/21)

Tasks:

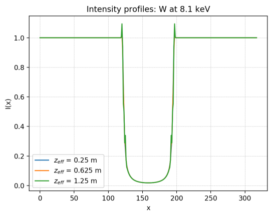
1. Meetings on **Monday 04/10** and **Wednesday 06/10**
   a. ~~Group: **2pm**~~
   b. ~~one on one: **11 am**~~
2. ~~*Are we likely to see fringes shift in MK's lab?*~~
3. Write report <<in progress>>
4. Prepare for oral presentation <<in progress>>

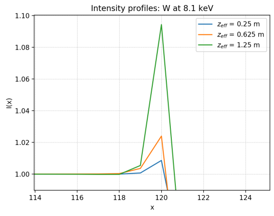
04/10/2021

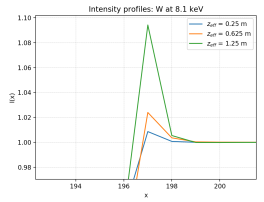
Are we likely to see phase contrast fringes shift in the Lab?

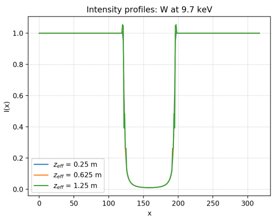
~~**TIE+RK**~~	~~M = 4.0X~~	
		
		
		
		
~~I am not super sure if we can observe fringes shifting from one energy to another...~~	~~It does look like the fringes shift from pixel **120** to **121** between E = 11.2 keV and E = 21 keV~~	~~And the fringes shift from pixel **120** to **121** between E = 11.2 keV and E = 21 keV~~

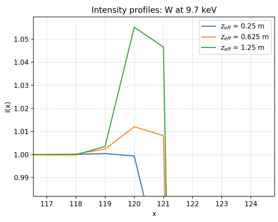
~~**AS**~~ ~~M = 4.0X~~

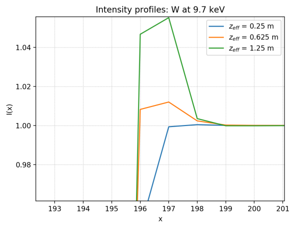


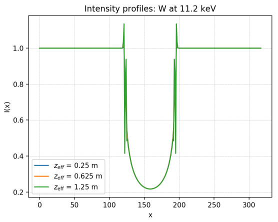


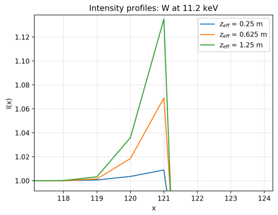


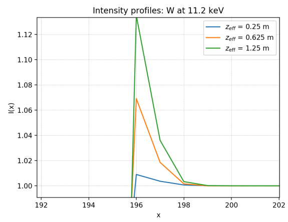


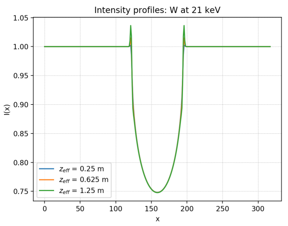


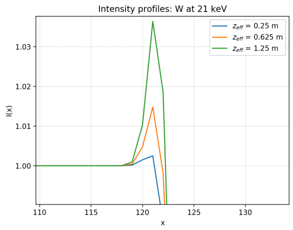


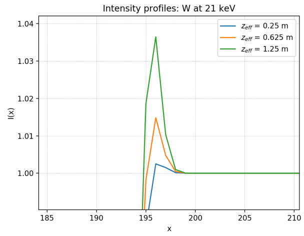












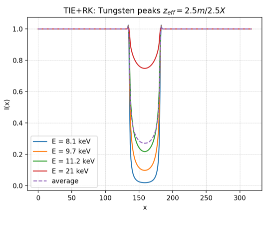
~~Yeah it looks like the fringes shift by one pixel from one energy to another~~

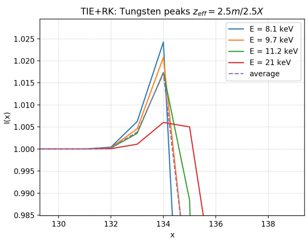
~~The TIE+RK outputs appear more stable and robust for the lab settings.~~

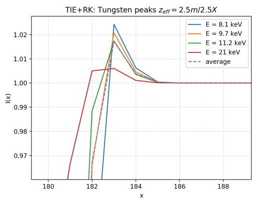
06/10/21

Maybe I can visualise this better by plotting all the energies for a specific distance

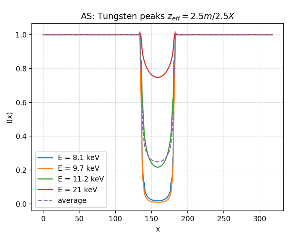
~~**TIE+RK**~~ ~~**M = 2.5X**~~

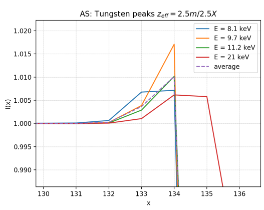


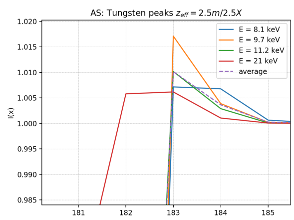




~~**AS**~~ ~~**M = 2.5X**~~

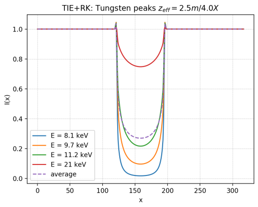


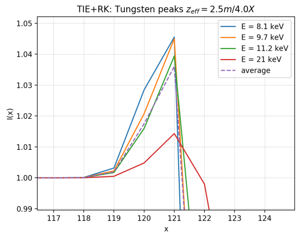


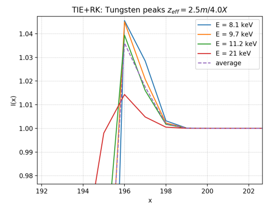


~~**Different magnification**~~

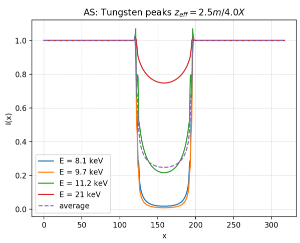
~~TIE+RK~~ ~~M = 4.0X~~

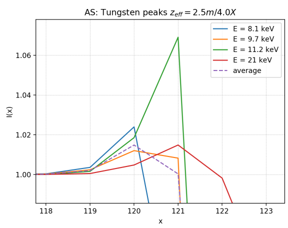


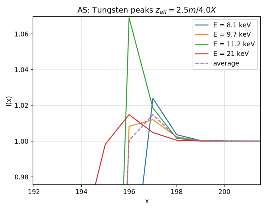




~~AS~~ ~~M = 4.0X~~



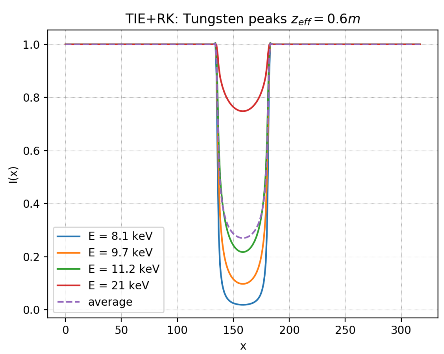


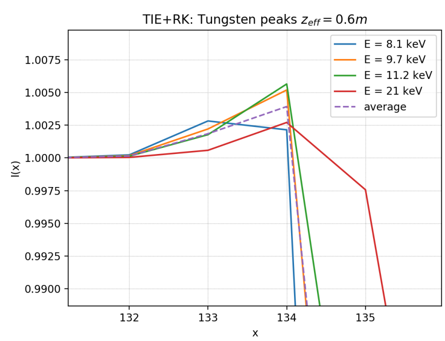


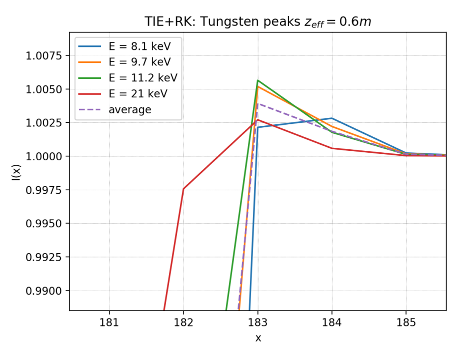
Whoops! MK informed me that my calculation of z_eff was incorrect!
This is the correct way to calculate it:

z_actual = 5 * m
z_final = (z_actual - (z_actual / M)) / M # eff propagation distance

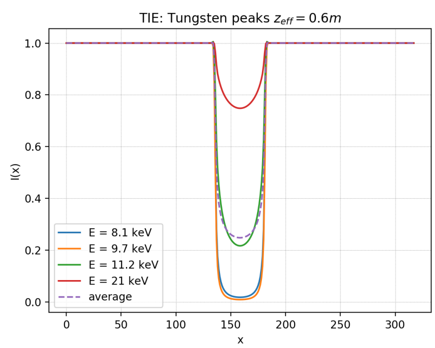
TIE+RK M = 2.5X

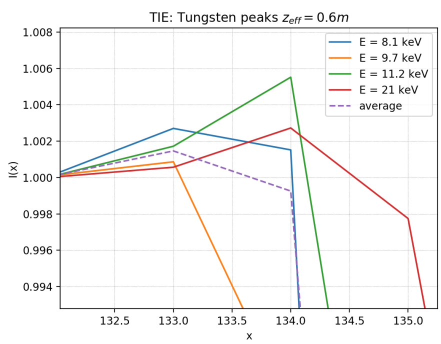


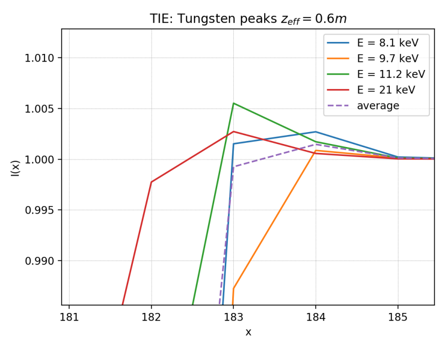




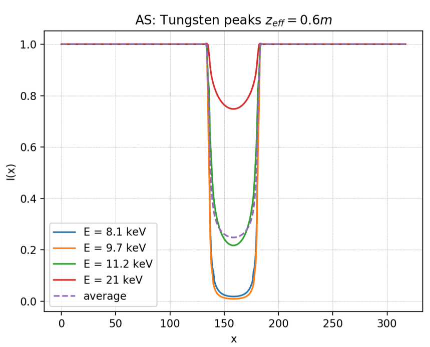
TIE M = 2.5X

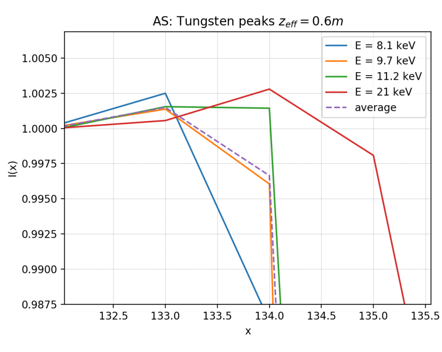


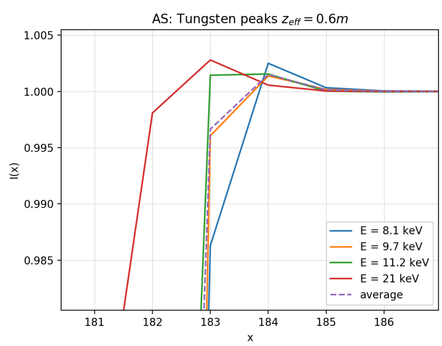




AS M = 2.5X

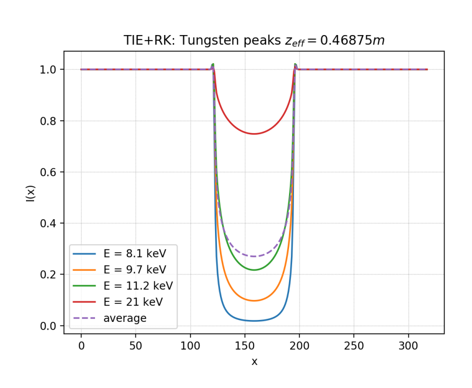


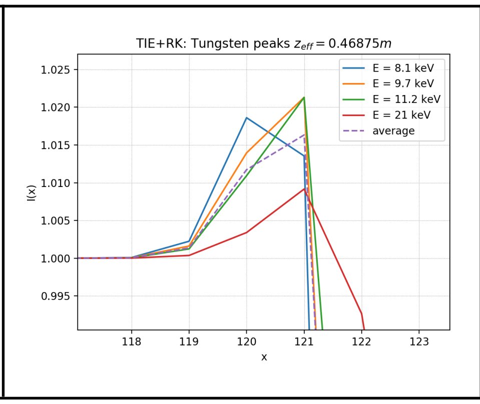


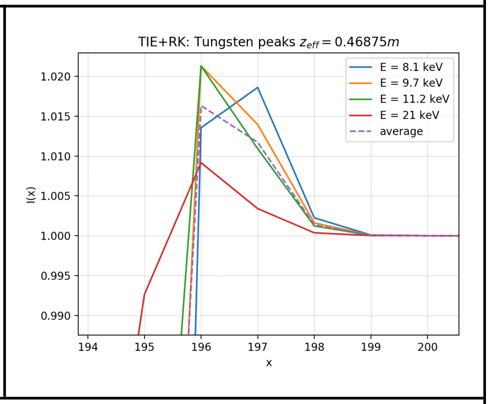


TIE+RK

M = 4.0X

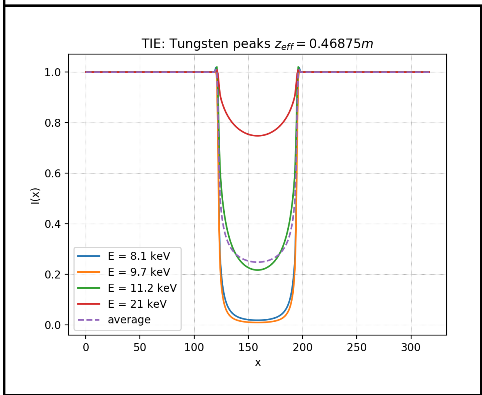


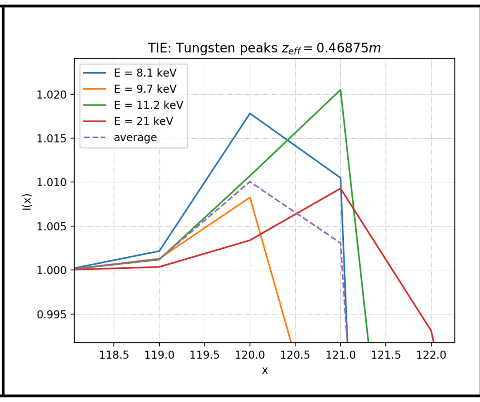


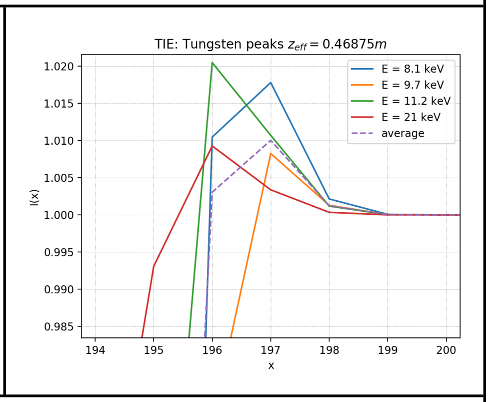


TIE

M = 4.0X

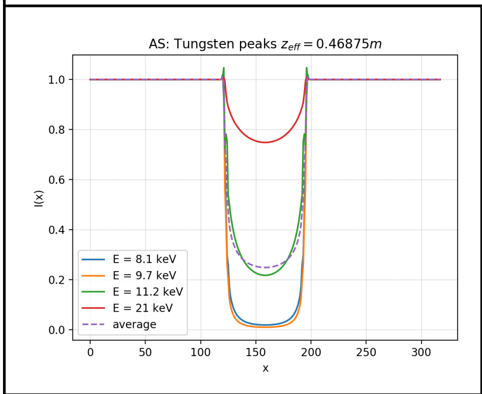


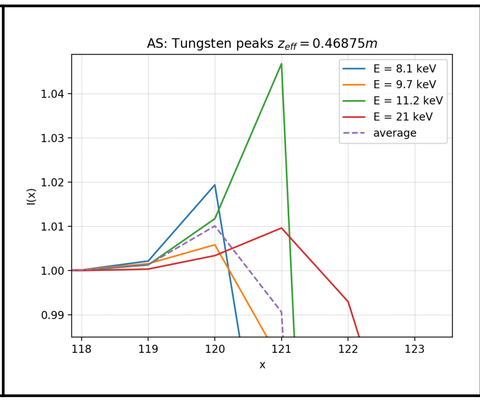


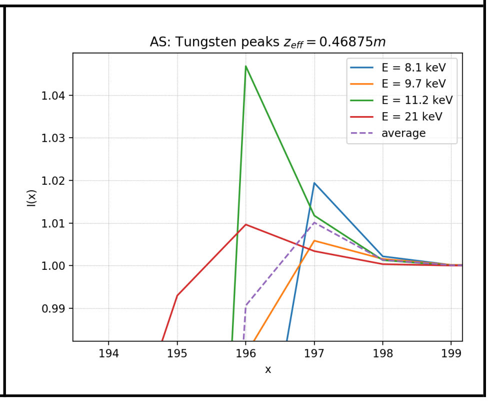


AS

M = 4.0X







From these results I think that we can possibly see the fringes shift in the lab. **We should test this.**
I'm only just noticing that the orange plot corresponding to E = 9.7keV is not as dark for TIE+RK as it is for AS and TIE… I've recalculated and the results for all methods with this energy and the outcome is still this way.

08/10/2021

LAB DAY

Because of COVID restrictions, MK & I did my experiment via zoom, with the help of JP.
Here are my notes:
Warming up the tube --runs through the voltage ranges max of 70kV
We are using the photon counting detector dimensions with 70mm×14mm, pixel size: 55um
Perspex rods: D~8mm <-MK will confirm this for me.

We are using the **TUNGSTEN** TARGET
Threshold set to: 12.5keV
(i.e. NO PHOTONS detected below 12.5keV in first run)

Every detector except for this one has an offset to minimise dark noise & dark current from thermal effects
Our detector doesn't have this problem

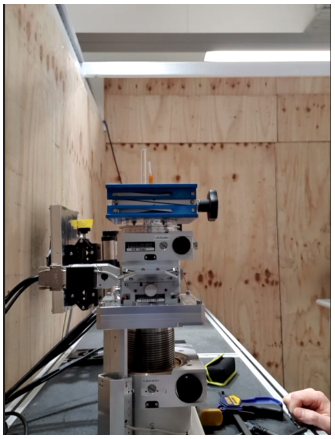
Flatfield looks like it has gaps between modules and a sort of imperfection pattern in the lower left corner. There also is a grid like pattern on the 2 left modules that MK is unsure about so he is doing a visual check. There was a plastic cover over the detector, it is now removed.

We have multiple rods:
We will only test **Rod2, and Rod4**

Rod1 D = 7.9mm	**Rod2** **D=9.8**	Rod3 D=7.9mm	**Rod4** **D=6.23mm** **Has a little hole drilled into it with D~2mm**, the hole doesn't go all the way along the rod and it is perpendicular to the beam so it shouldn't affect the beam propagation

MAIN IDEA:
How does the energy spectrum affect phase contrast, does imaging using the whole spectrum blurr the results?
The simulations show that all the CR peaks produce PC at a different pixel than

The set up:



The propagation distance we will test is **SDD**: 2.501m (as measured with a laser meter)
We will use a magnification of 2.5X
And a magnification of 4.0X

Which gives
Z_eff = 0.6
Z_eff = 0.47

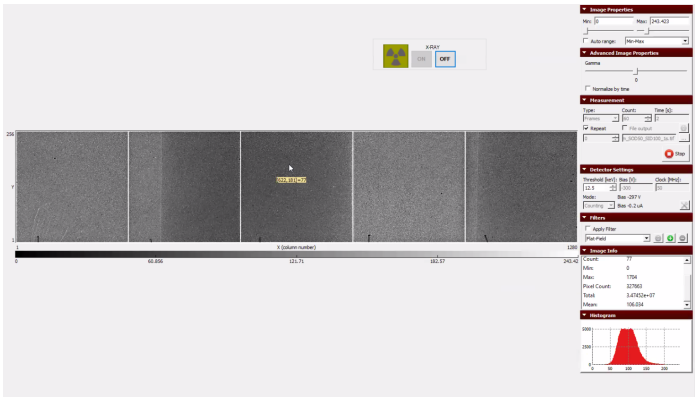
SOD: 1m

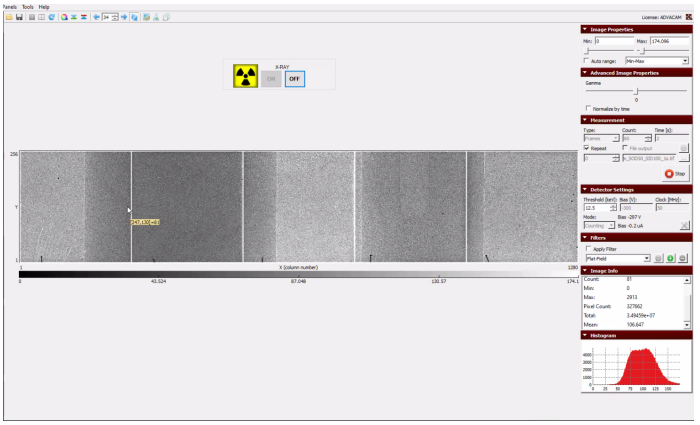
TARGET POWER IS UP TO 20W

The more power the beam has the less coherence we see

We have no horizontal control at the moment, so we has do adjust the horizontal location of the beams

FIRST IMAGE TEST OF RODS positioning:





We want 5 images for each threshold @ 20s exposure each

DATA FOLDERS:
delta_2021-10-08_Fabela_energyTest
SOD100cm
SOD62p5cm

2.5X	**4.0X**
::::Imaging the flatfield::::: Flats_35kV_20W_SOD100_SID250cm_20s_7keVThreshold **~750 counts avg** Flats_35kV_20W_SOD100_SID250cm_20s_12p5keVThreshold The number of counts decreased, looks like the spectrum is dominated by low energy photons **~540 counts** Flats_35kV_20W_SOD100_SID250cm_20s_19keVThreshold **~150 counts** **::::Imaging the rods:::::** Rods_35kV_20W_SOD100_SID250cm_20s_7keVThreshold Rods_35kV_20W_SOD100_SID250cm_20s_12p5keVThreshold Rods_35kV_20W_SOD100_SID250cm_20s_19keVThreshold	**::::Imaging the flatfield:::::** Flats_35kV_20W_SOD62p5cm_SID250cm_20s_7keVThreshold **~ 1600 counts** Flats_35kV_20W_SOD62p5cm_SID250cm_20s_12p5keVThreshold **~ 600 counts** Flats_35kV_20W_SOD62p5cm_SID250cm_20s_19keVThreshold **~100 counts** **::::Imaging the rods:::::** Rods_35kV_20W_SOD62p5cm_SID250cm_20s_7keVThreshold Rods_35kV_20W_SOD62p5cm_SID250cm_20s_12p5keVThreshold Rods_35kV_20W_SOD62p5cm_SID250cm_20s_19keVThreshold

More propagation distance with lower magnification but more resolution with higher magnification

We are taking more data now we are only using Rod2 since it looks like Rod4 was moving (falling sideways towards Rod2)

We are doing 10 exposures of each threshold

::::Imaging Rod2:::::
Rod_35kV_20W_SOD100_SID250cm_20s_7keVThreshold
Rod_35kV_20W_SOD100_SID250cm_20s_12p5keVThreshold
Rod_35kV_20W_SOD100_SID250cm_20s_19keVThreshold

DATA ANALYSIS
TODO:

1. ~~Open flats for **each energy threshold**~~
    a. ~~Flats image averaging~~
2. ~~FLATFIELD CORRECTION (RODS/FLATFIELD)~~

10/10/2021

Working on cleaner DATA for report 3

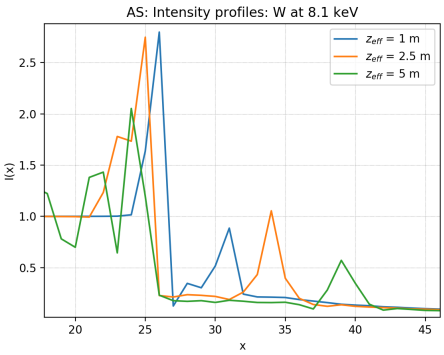
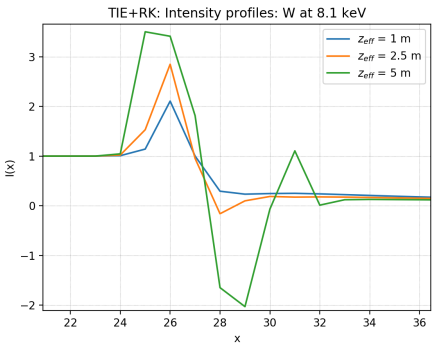
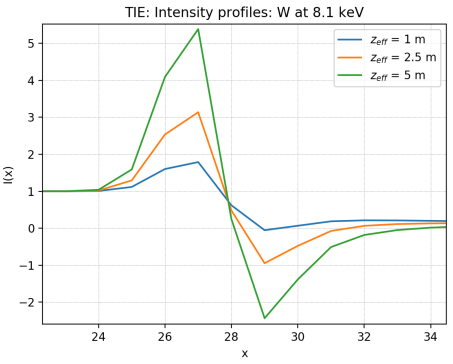
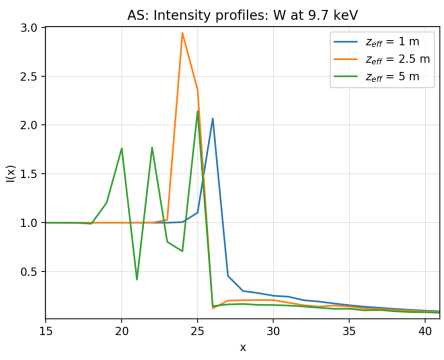
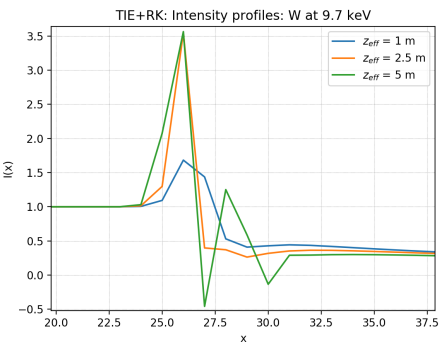
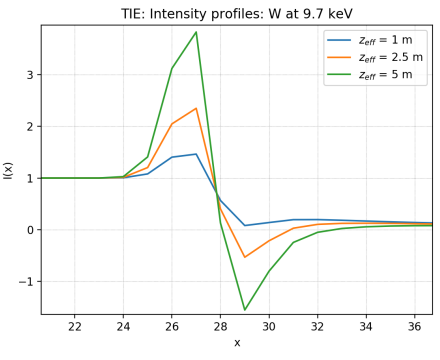
I've been revising my simulations to gather results and write my report.

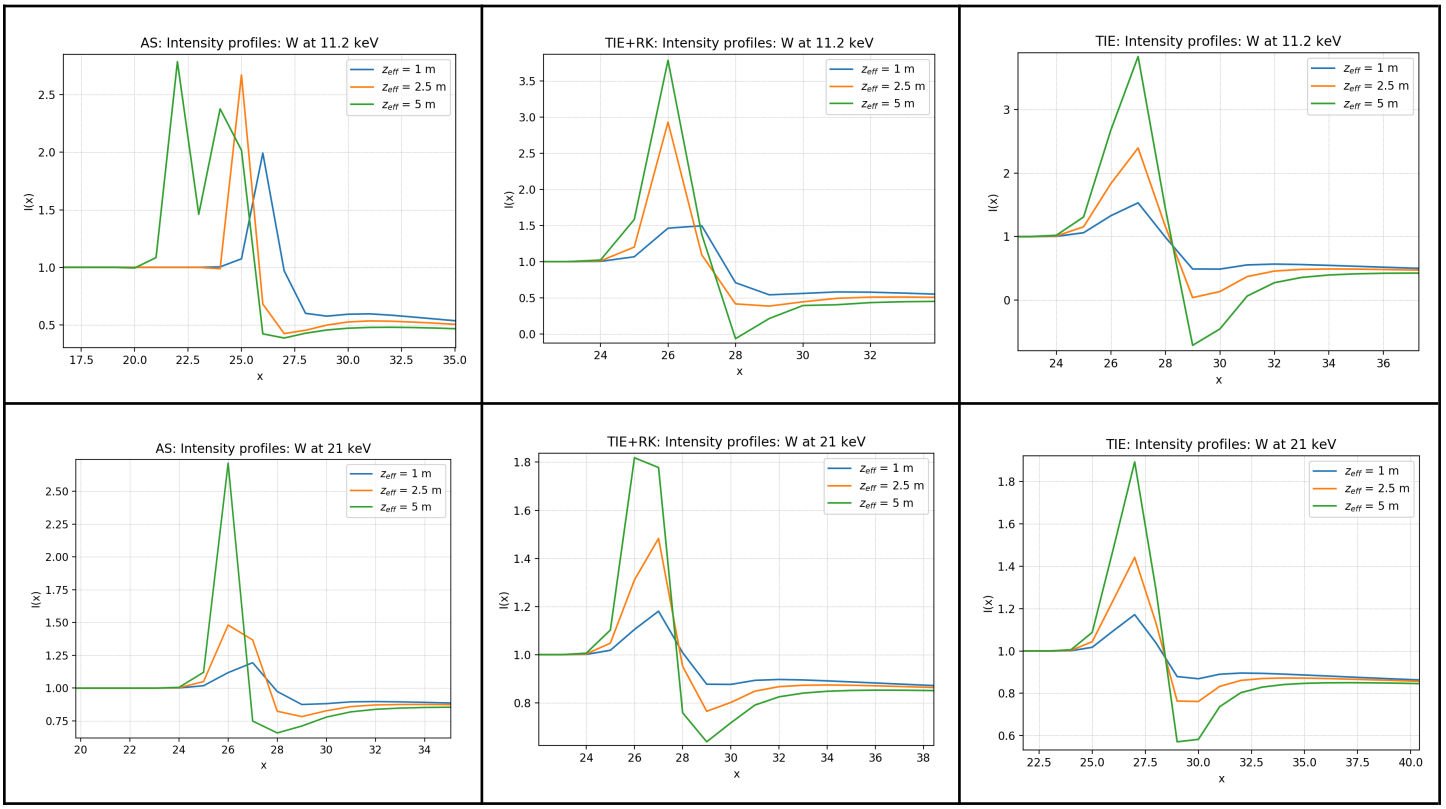
One of the tests that MK suggested involved verifying **if** the TIE+RK showed that the fringe position will shift with **Energy** (like the AS does). The comparison appears logical since according to MK the **regular TIE fringes never budge.**
I am checking this with resampled data (with data that doesn't replicate the Lab parameters, and with data that does replicate Lab parameters).

In the two tables below I am testing all the energy peaks for the Tungsten spectrum at 35 kV individually for each method (i.e. AS, TIE+RK, TIE).

1. So far it looks like the TIE+RK is less susceptible to ringing artifacts than the AS.
2. The regular TIE and the TIE+RK also appear to go below zero a few times while the AS does not.
3. It is hard to tell if the TIE+RK is more accurate than the regular TIE at this point.

The figures in this table should be observed and compared per column from top to bottom.

IDEAL DATA	(ONLY LEFT FRINGES)	
AS: fringes move from **pxl 25 → 26**	**TIE+RK**: fringes move from **pxl 26 → 27**	**TIE**: fringes do not move from **pxl 27**
		
		

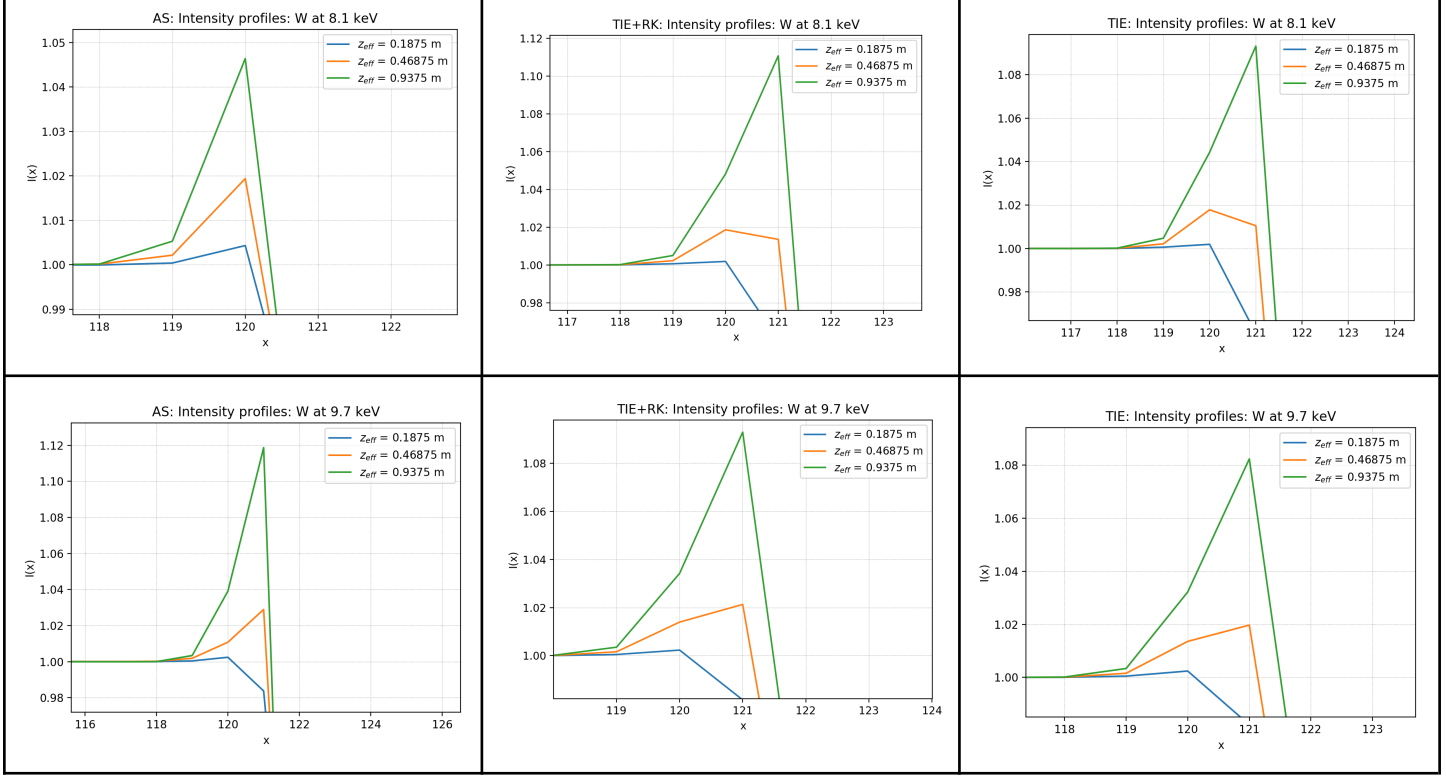


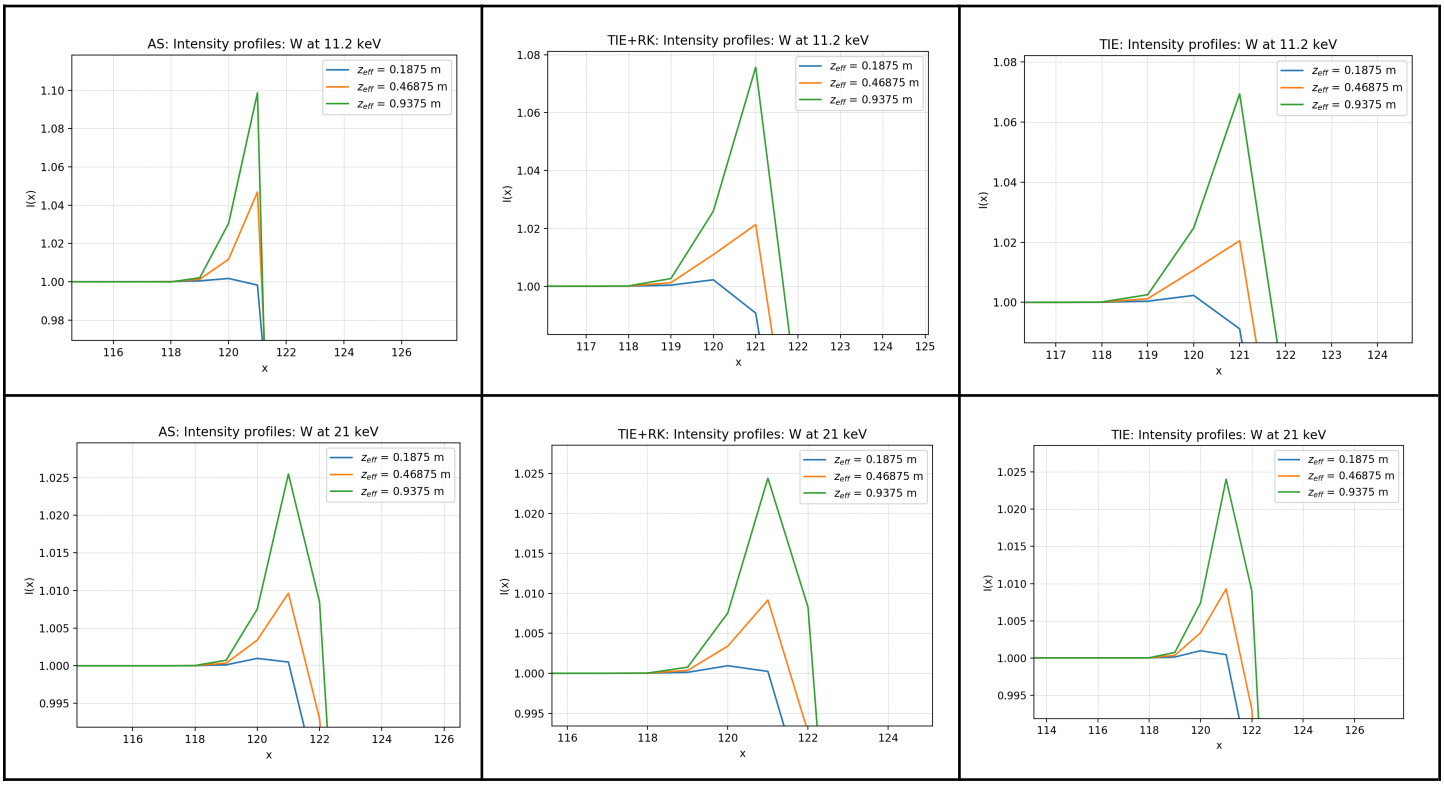
These are the left fringes from the Lab parameter simulations, we are testing these in the experiment (together with the 2.5X magnification which I have not included here).

1. The regular TIE and the TIE+RK difficult to interpret when compared to **AS**
2. It looks like neither the TIE and the TIE+RK shifts

Here there isn't a visible improvement between the TIE+RK and the TIE either since the fringes

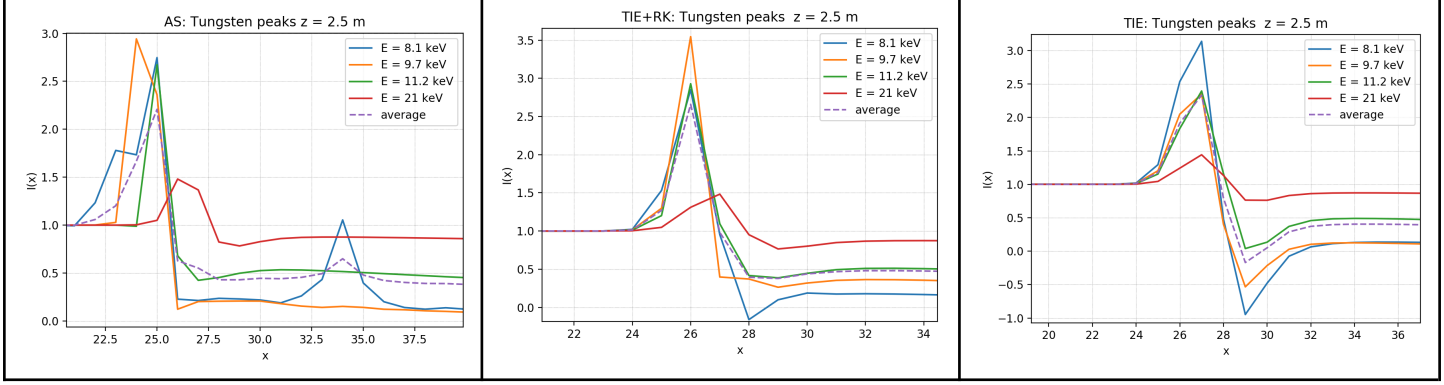
LAB DATA @ **M = 4.0X**	(ONLY LEFT FRINGES)	
AS: fringes move from **pxl 120 → 121**	TIE+RK: **not clear**	TIE: **not clear**





HERE it looks like only the AS and the TIE+RK predict a fringe shift

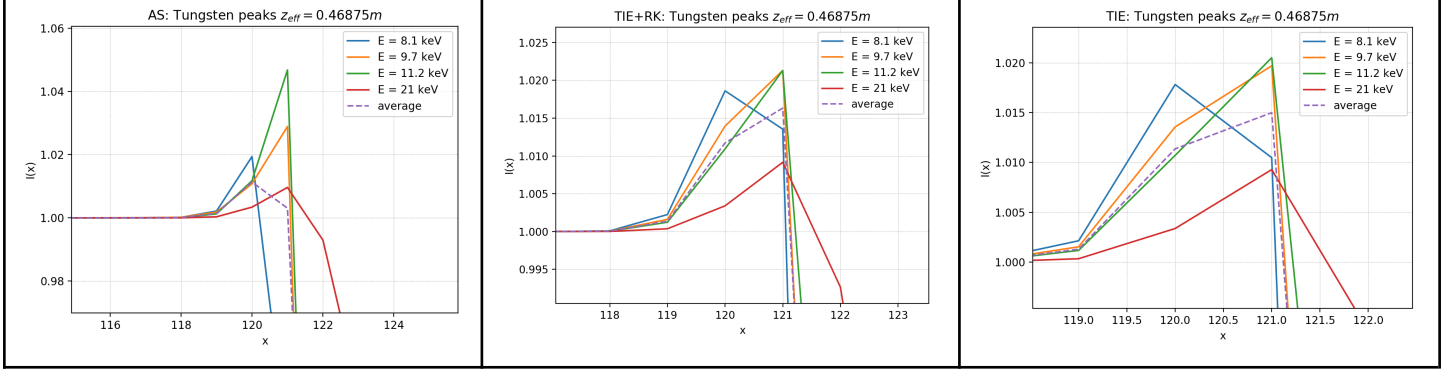
IDEAL DATA	(ONLY LEFT FRINGES)	
AS	TIE+RK	TIE

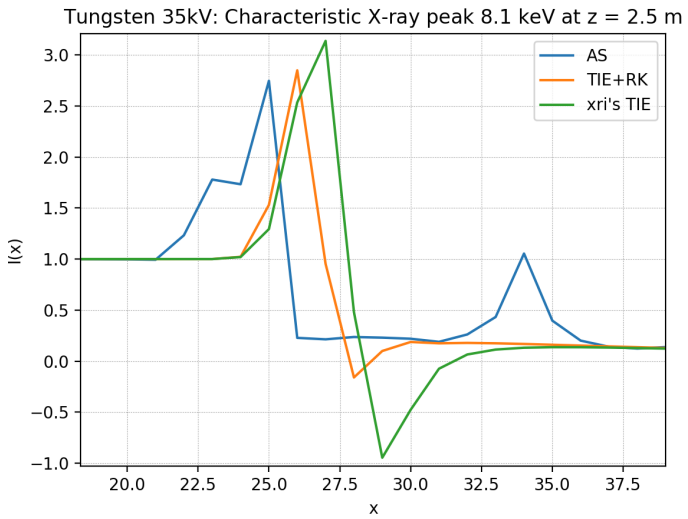
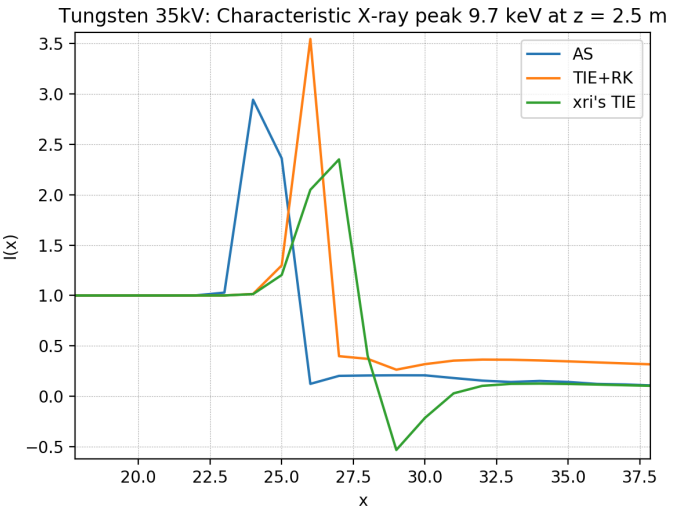
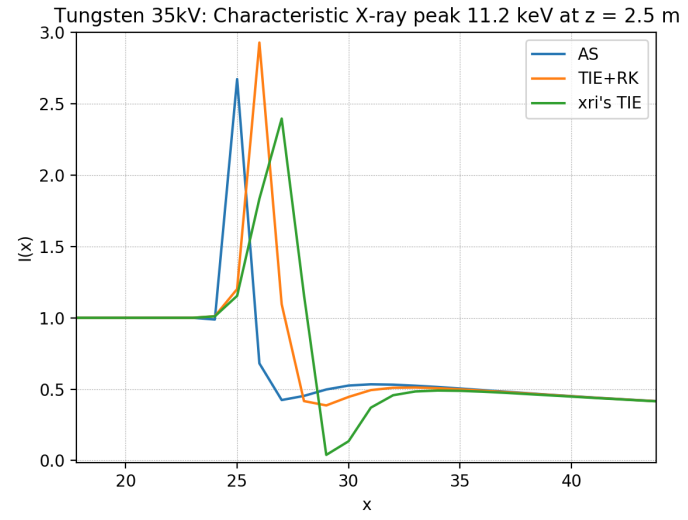
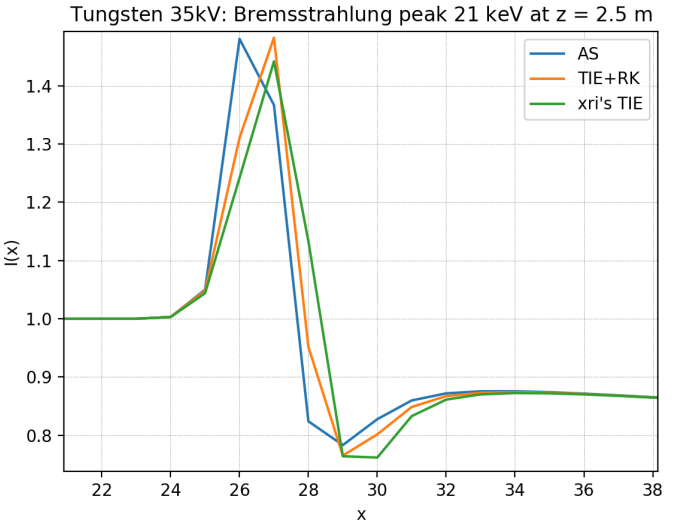


While using the “Lab” parameters it looks like all methods apparently predict a shift?

...THIS IS SO CONFUSING and it does not match the expectation of the TIE at all

LAB DATA @ **M = 4.0X**	(ONLY LEFT FRINGES)	
AS	TIE+RK	TIE

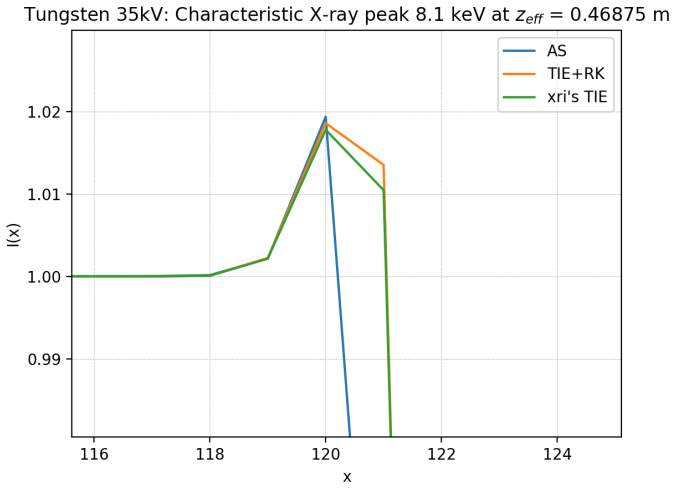


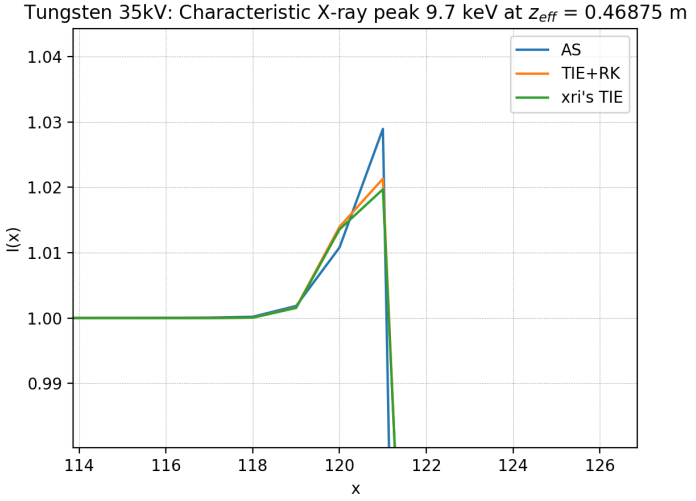
IDEAL DATA	(ONLY LEFT FRINGES)
8.1 keV **AS: pxl = 25** **TIE+RK: pxl = 26** **TIE: pxl = 27**	9.7 keV **AS: pxl = 24** **TIE+RK: pxl = 26** **TIE: pxl = 27**
	
11.2 keV **AS: pxl = 25** **TIE+RK: pxl = 26** **TIE: pxl = 27**	21 keV **AS: pxl = 26** **TIE+RK: pxl = 27** **TIE: pxl = 27**
	

Both AS and TIE+RK fringes appear to shift by one pixel from the 11.2keV peak to the Bremsstrahlung peak 21 keV
While the TIE fringe does not move

THE **AS** FRINGES ARE A LOT MORE “RINGY” THOUGH SO IT’S A BIT HARD TO KNOW WHAT'S UP

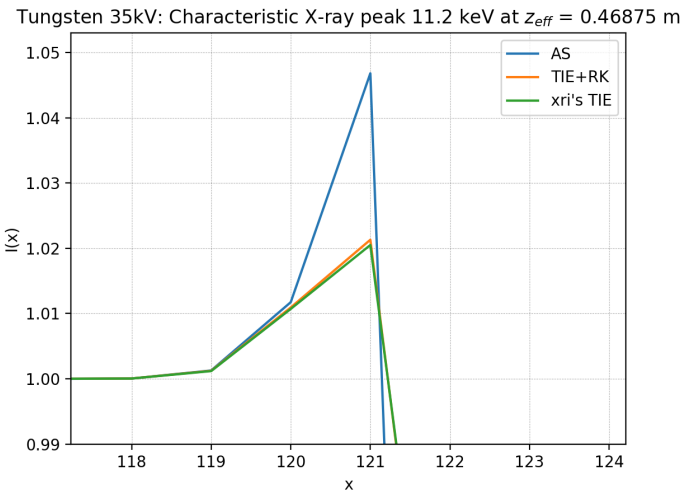
LAB DATA @ **M = 4.0X**	(ONLY LEFT FRINGES)
8.1 keV **AS: pxl = 120** **TIE+RK: pxl = 120** **TIE: pxl = 120**	9.7 keV **AS: pxl = 121** **TIE+RK: pxl = 121** **TIE: pxl = 121**

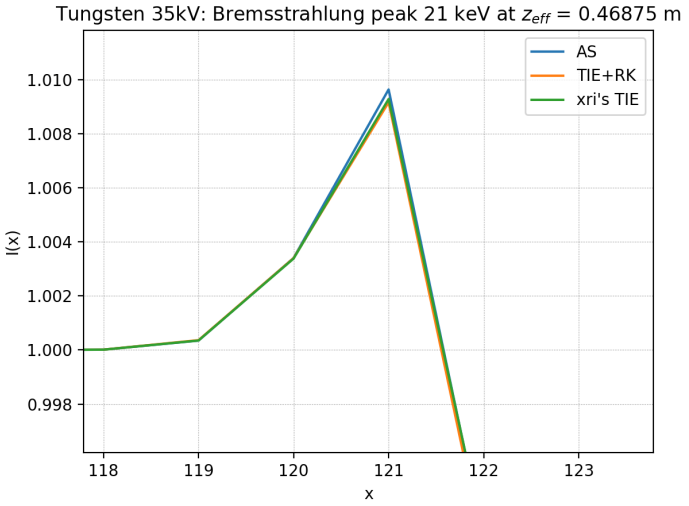




11.2 keV
AS: pxl = 121
TIE+RK: pxl = 121
TIE: pxl = 121

21 keV
AS: pxl = 121
TIE+RK: pxl = 121
TIE: pxl = 121



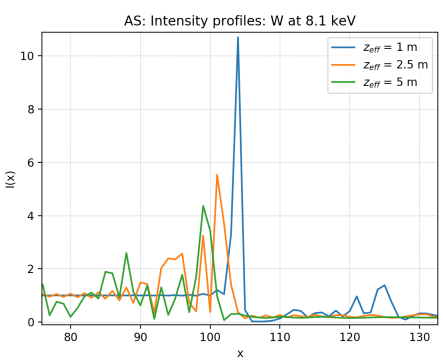
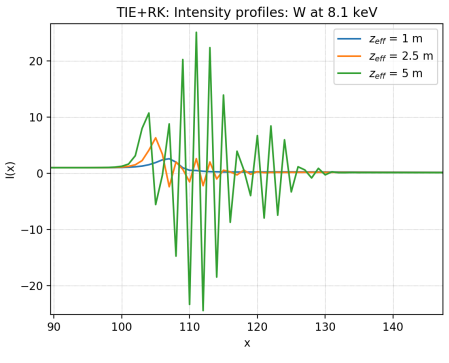
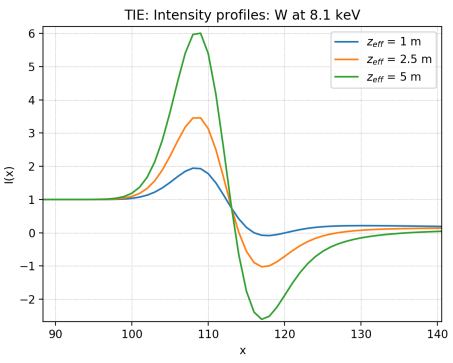


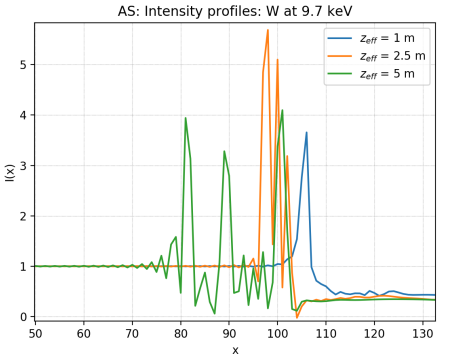
HERE THEY ALL SHIFT BY ONE PIXEL!
WHAT IS HAPPENING?!?!?

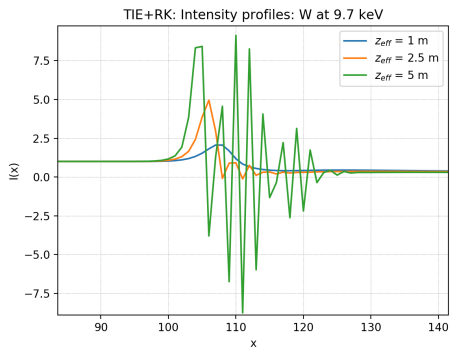
So maybe the issues are due to the resampling… I should test all the ideal cases again **without resampling.**

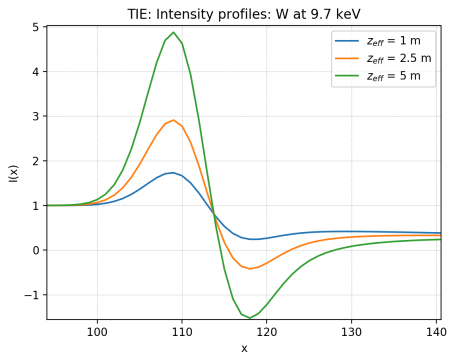
IDEAL DATA - no resampling

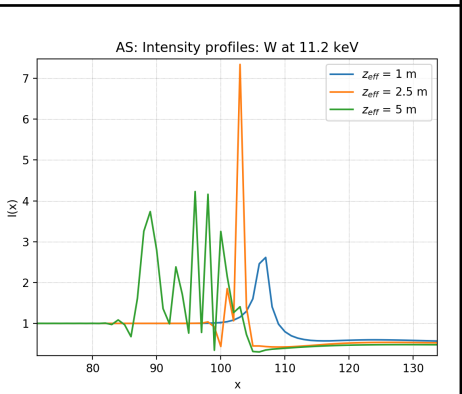
The figures in this table should be observed and compared per column from top to bottom.

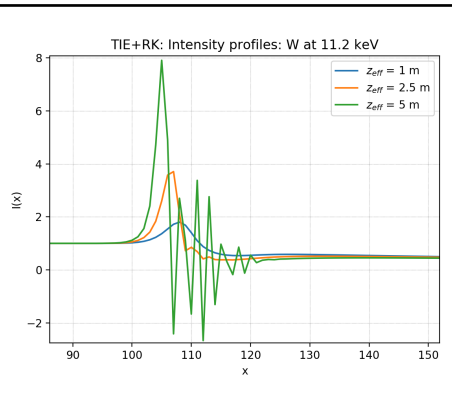
IDEAL DATA	(ONLY LEFT FRINGES)	
AS: from (8.1 → 21) keV 1 m: fringes move from **pxl 104→ 109** 2.5 m: fringes move from **pxl 101→ 107** 5 m: fringes move from **pxl 99 → 105**	TIE+RK: 1 m: fringes move from **pxl 107 → 109** 2.5 m: fringes move from **pxl 105→ 109** 5 m: fringes move from **pxl 104 → 108**	TIE: fringes DO NOT move from **pxl 109**
		

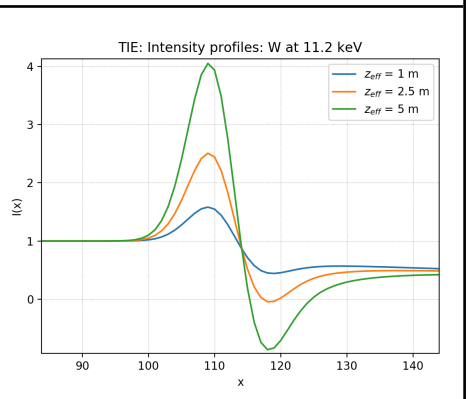


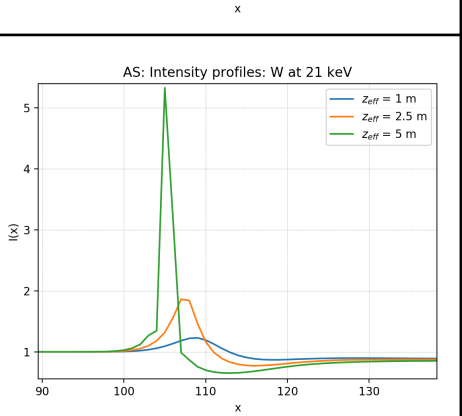


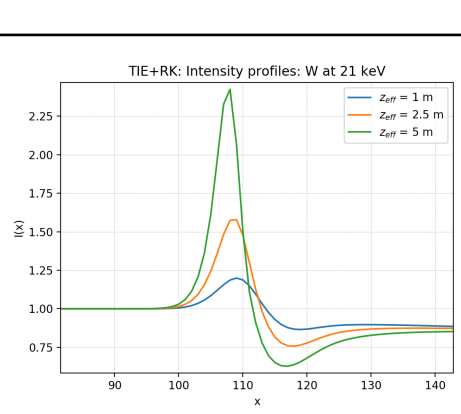


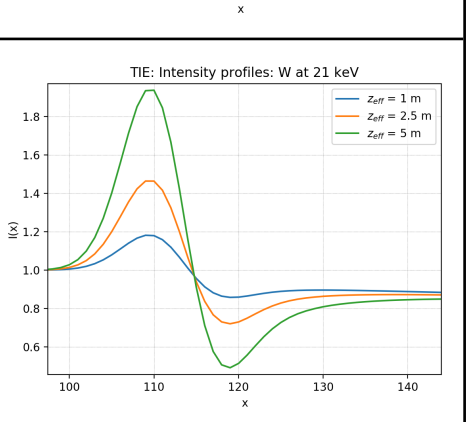




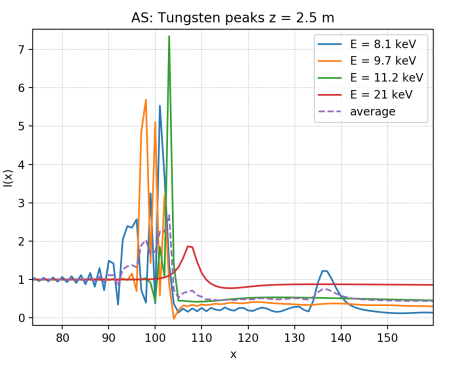


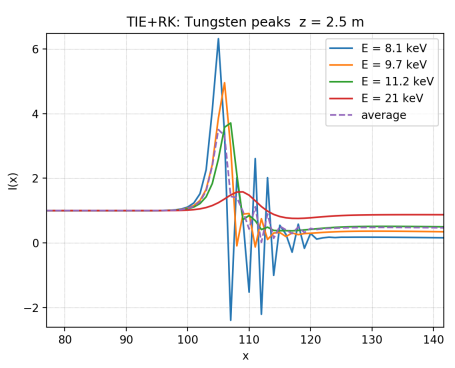


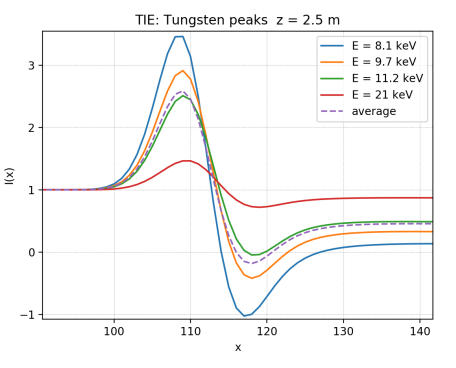




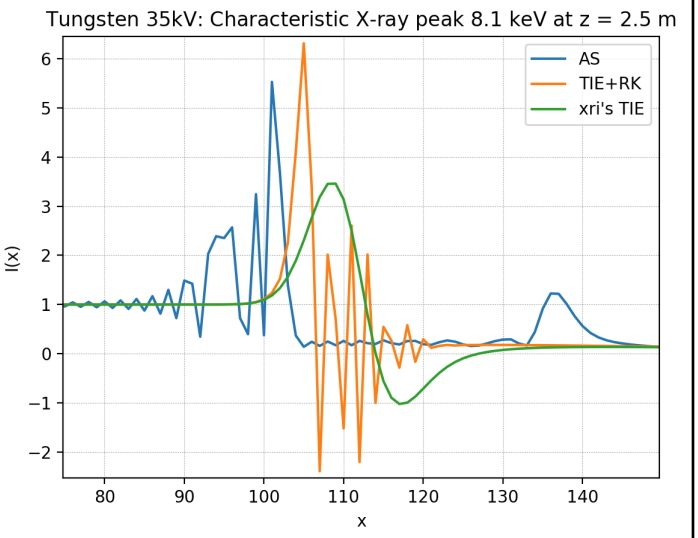
IDEAL DATA	(ONLY LEFT FRINGES)	
AS	TIE+RK	TIE

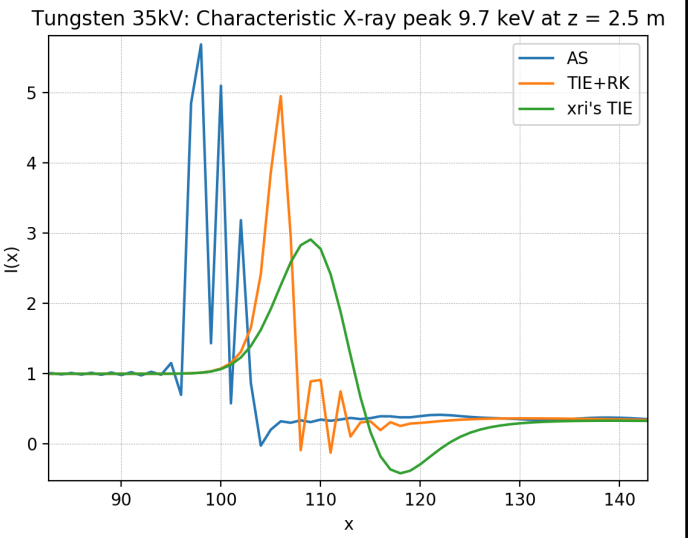






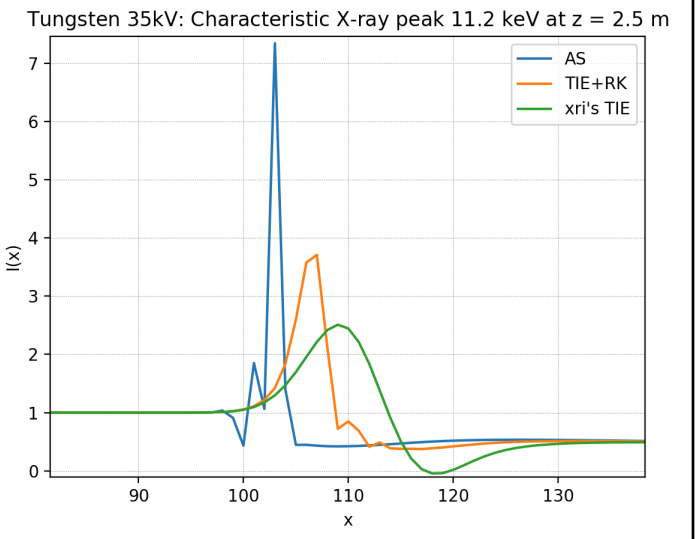
IDEAL DATA	(ONLY LEFT FRINGES)
8.1 keV **AS: pxl = 101** **TIE+RK: pxl = 105** **TIE: pxl = 109**	9.7 keV **AS: pxl = 102** **TIE+RK: pxl = 106** **TIE: pxl = 109**

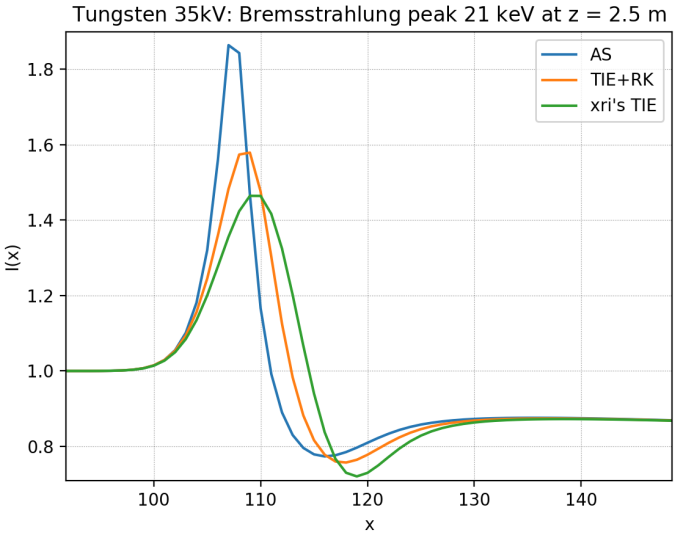




11.2 keV
AS: pxl = 103
TIE+RK: pxl = 107
TIE: pxl = 109

21 keV
AS: pxl = 107
TIE+RK: pxl = 109
TIE: pxl = 109





IN CONCLUSION: in the most ideal circumstances it appears that the TIE+RK is the most similar to the AS while at the same time being slightly more stable than the AS itself.
Something worth mentioning are the remaining “ringing” intensity fringes that accompany the main phase contrast fringe in the TIE+RK… are always towards the inner part of the imaged cylinder. I saw similar behaviour during my experiment so I am looking forward to checking if these are correctly being predicted by the TIE+RK while not being predicted at all by the TIE.

Week 11 (11/10/21 –17/10/21)

Tasks:

1. Meetings on **Monday 11/10** and **Wednesday 13/10**
    a. ~~Group: **2pm**~~
    b. ~~one-on-one: **11 am**~~
2. Write report
3. Prepare for oral presentation
4. Continue LAB analysis
5. Make dot points for presentation & send to MK

11/10/2021

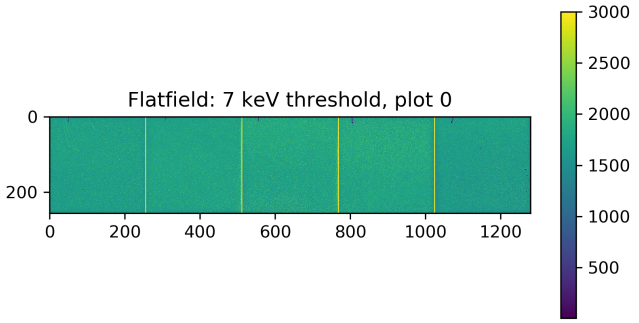
DATA ANALYSIS

Separating the Flats from the Rods data sets

```python
# folder = Path('SOD62p5cm/')
folder = Path('SOD100cm/')


Flats = []
Rods = []
single_Rod = []
for p in folder.iterdir():
    if 'Flats' in p.name:
        Flats.append(p)
    if 'Rods' in p.name:
        Rods.append(p)
    else:
        single_Rod.append(p)

np_Flats_7keV = []
np_Flats_12p5keV = []
np_Flats_19keV = []
for p in Flats:
    try:
        # print(im.mode)
        np_a = np.array(Image.open(p))
    except:
        raise
    else:
        if '7keV' in p.name:
            np_Flats_7keV.append(np_a)
        elif '12p5keV' in p.name:
            np_Flats_12p5keV.append(np_a)
        else:
            np_Flats_19keV.append(np_a)

```

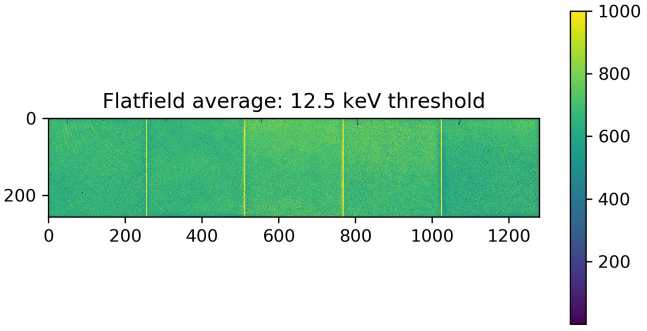
Plot Flats for each energy threshold	Clipping vmax = 3000 Example plot of Flatfield at 7keV

```python
for i, array in enumerate(np_Flats_7keV):
    print(f"{array.min()}, {array.max()}, {array.mean()}, {array.std()}")
    c = plt.imshow(array)
    plt.colorbar(c)
    plt.title(f"Flatfield: 7 keV threshold, plot {i}")
    plt.show()

for i, array in enumerate(np_Flats_12p5keV):
    print(f"{array.min()}, {array.max()}, {array.mean()}, {array.std()}")
    c = plt.imshow(array)
    plt.colorbar(c)
    plt.title(f"Flatfield: 12.5 keV threshold, plot {i}")
    plt.show()

for i, array in enumerate(np_Flats_19keV):
    print(f"{array.min()}, {array.max()}, {array.mean()}, {array.std()}")
    c = plt.imshow(array)
    plt.colorbar(c)
    plt.title(f"Flatfield: 19 keV threshold, plot {i}")
    plt.show()

```



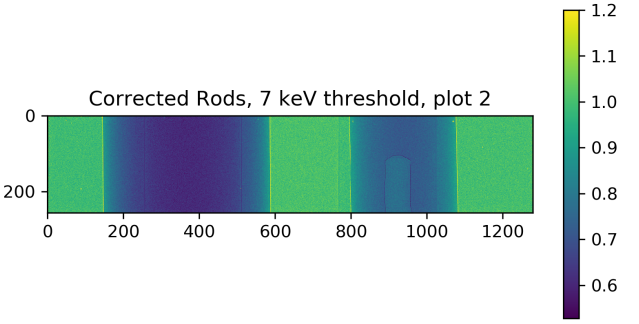
Average flatfield for each threshold

```
avg_Flats_7keV = sum(np_Flats_7keV) / len(np_Flats_7keV)

avg_Flats_12p5keV = sum(np_Flats_12p5keV) / len(np_Flats_12p5keV)

avg_Flats_19keV = sum(np_Flats_19keV) / len(np_Flats_19keV)
```




```python
corrected_np_Rods_7keV = []
corrected_np_Rods_12p5keV = []
corrected_np_Rods_19keV = []

for p in Rods:
    try:
        np_a = np.array(Image.open(p))
    except:
        raise
    else:
        if '7keV' in p.name:
            corrected_np_a = np_a / avg_Flats_7keV
            corrected_np_Rods_7keV.append(corrected_np_a)
        elif '12p5keV' in p.name:
            corrected_np_a = np_a / avg_Flats_12p5keV
            corrected_np_Rods_12p5keV.append(corrected_np_a)
        else:
            corrected_np_a = np_a / avg_Flats_19keV
            corrected_np_Rods_19keV.append(corrected_np_a)

print("TWO PERSPEX RODS -- IMAGES")

for j, array in enumerate(corrected_np_Rods_7keV):
    # print(f"{array.min()}, {array.max()}, {array.mean()}, {array.std()}")
    c = plt.imshow(array, vmax=1.2)
    plt.colorbar(c)
    plt.title(f"Corrected Rods, 7 keV threshold, plot {j}")
    plt.show()

```

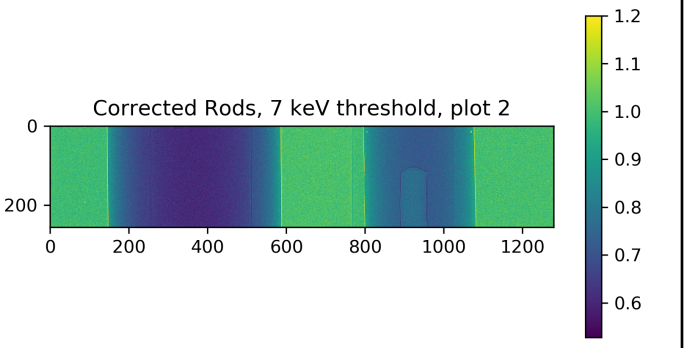
Example: Phase contrast Image of **Rod2** and **Rod4**

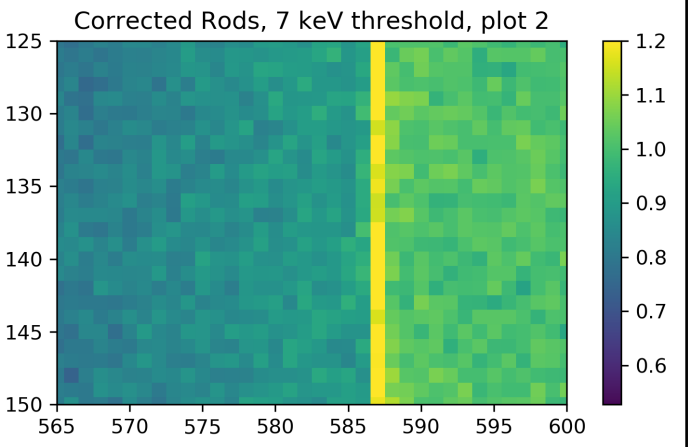


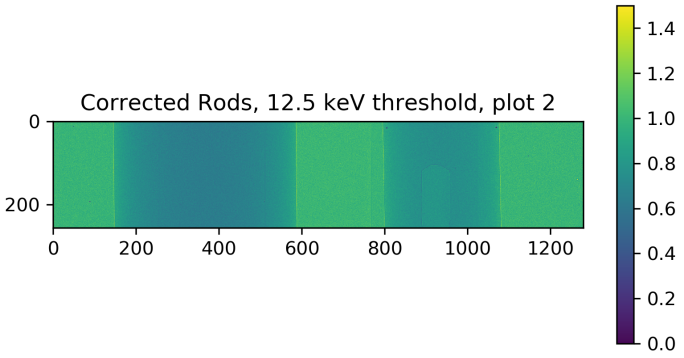
Corrected data -- SOD100cm

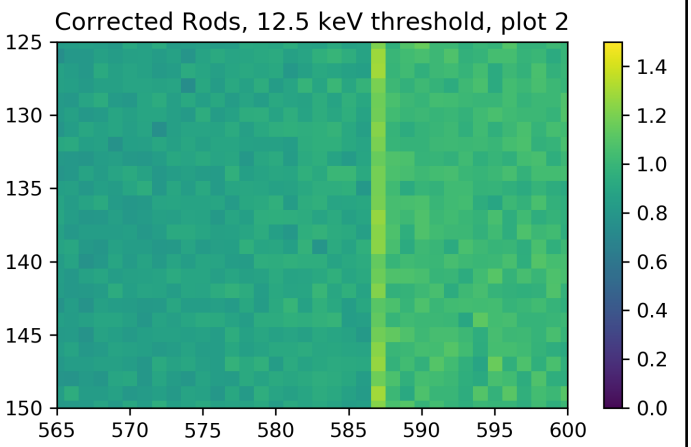
<<M = 2.5X>>

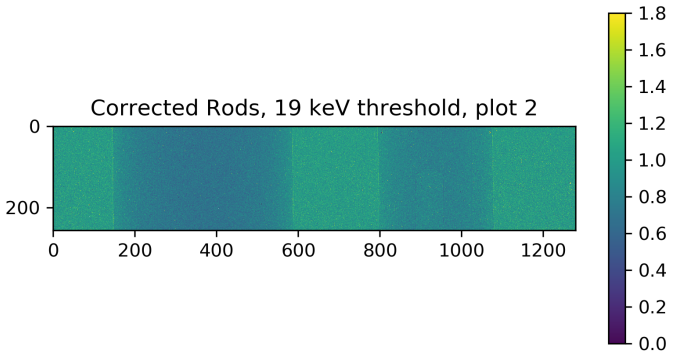
enhance!

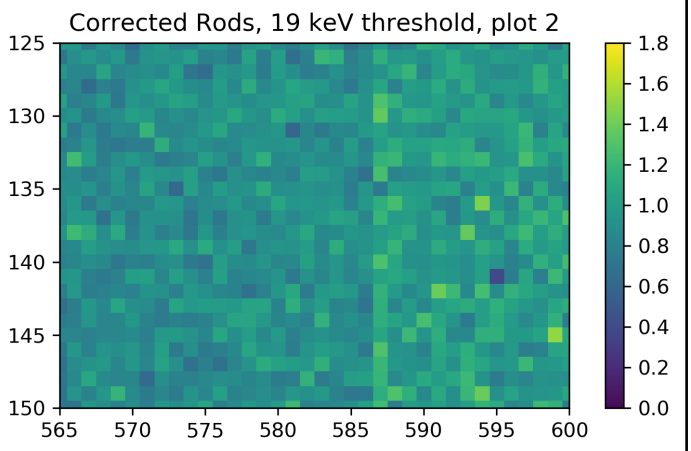








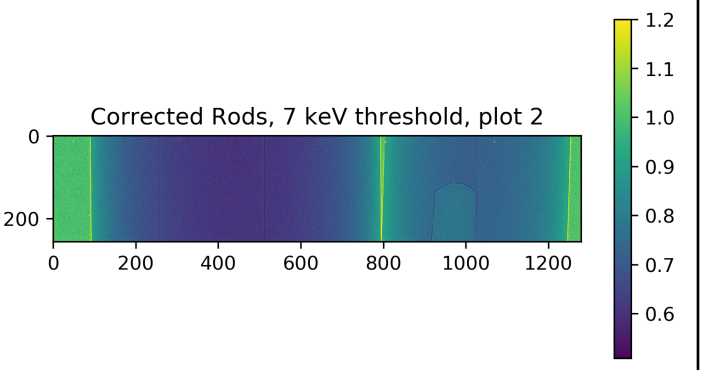


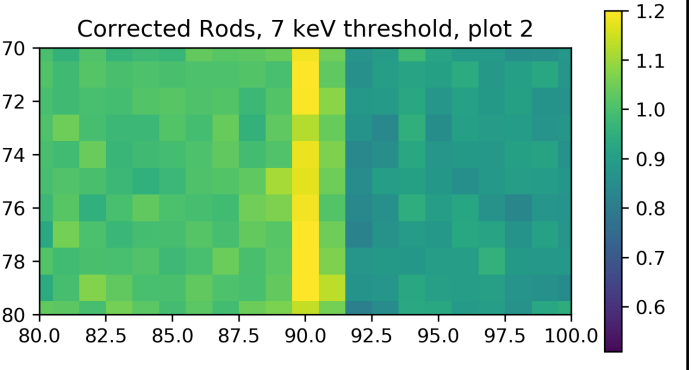


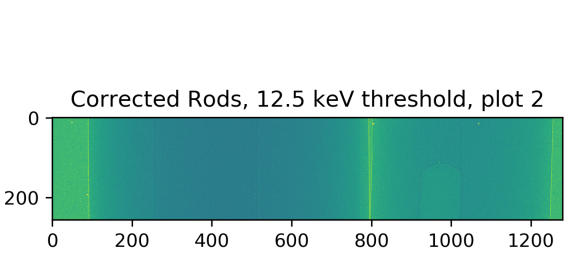
Corrected data -- SOD62p5cm
2Rods

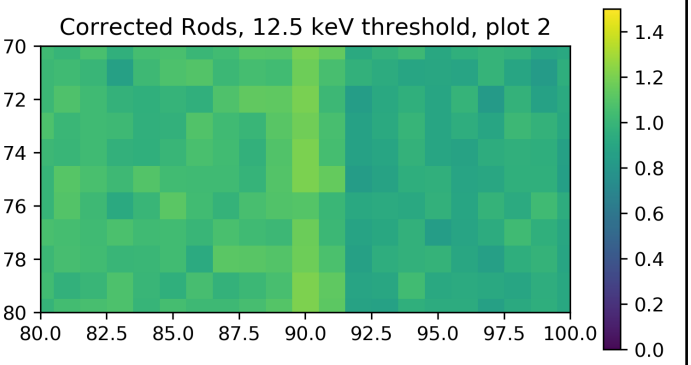
<<M = 4.0X>>

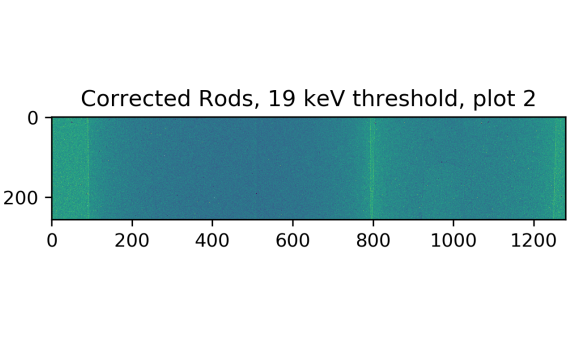
enhance!

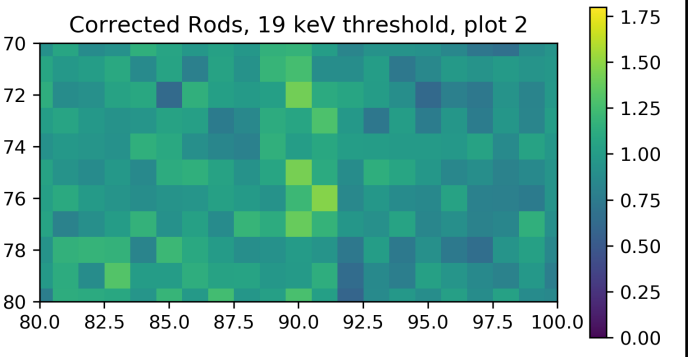








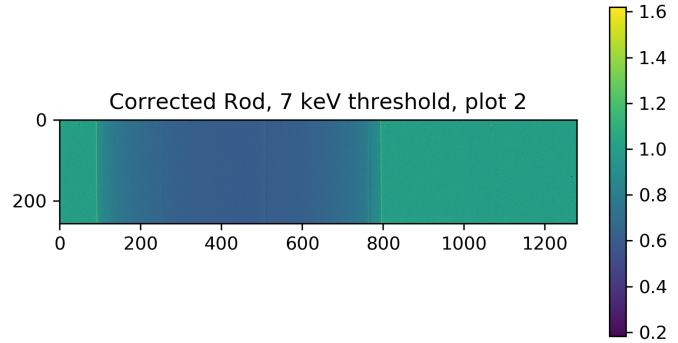


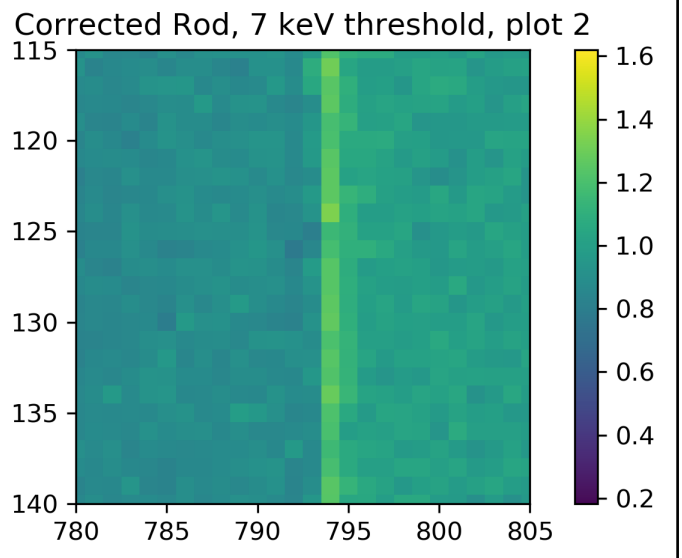


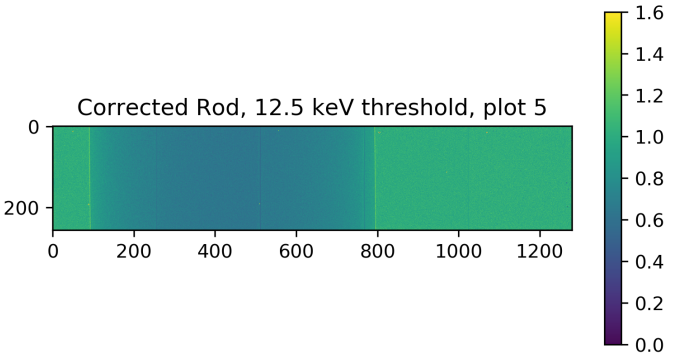
1Rod

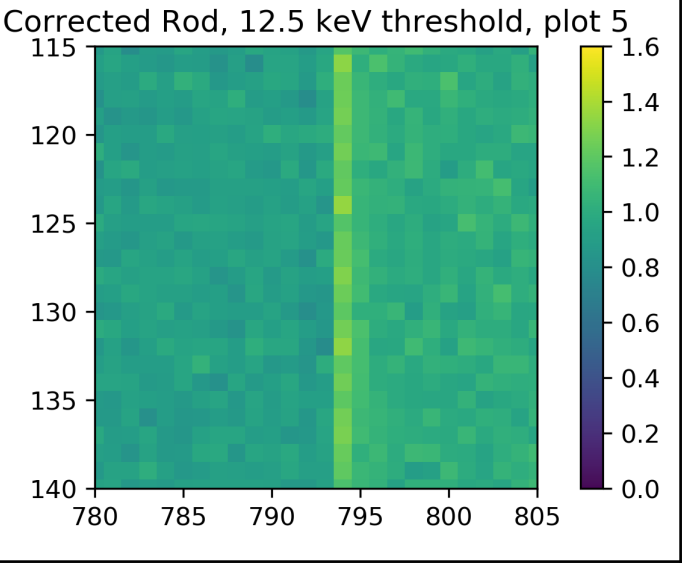
<<M = 4.0X>>

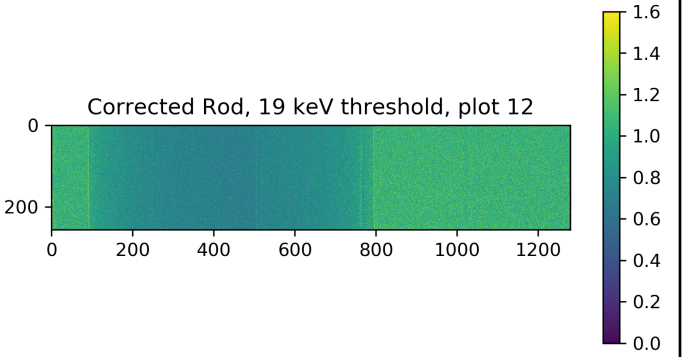
enhance!

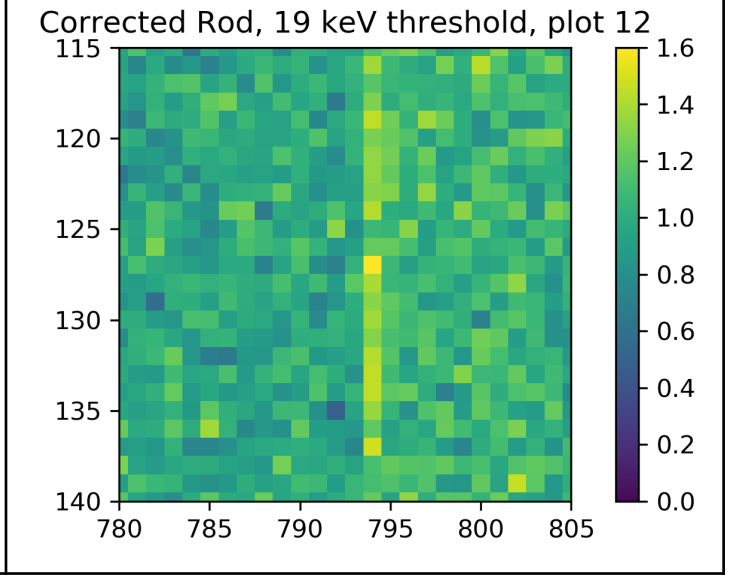












The fringe blurring looks much clearer with higher Magnification as expected.

12/10/2021

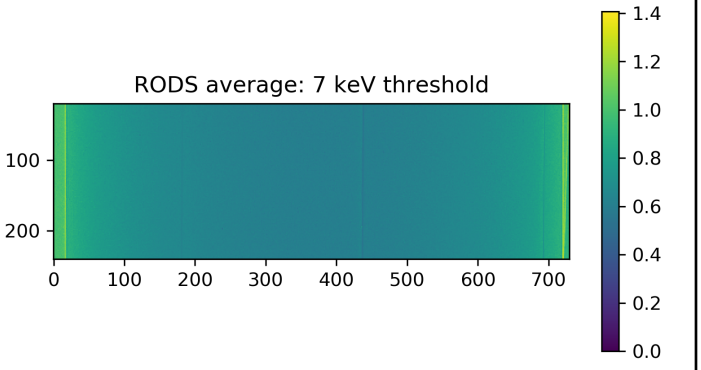
MK: *"It looks to me, from the last data set you showed, that the 19 keV threshold has a peak in only 1 pixel, whilst the other two thresholds look to have the peak spread over two pixels as we'd hoped. Please average the rows together to see if this is true. "*

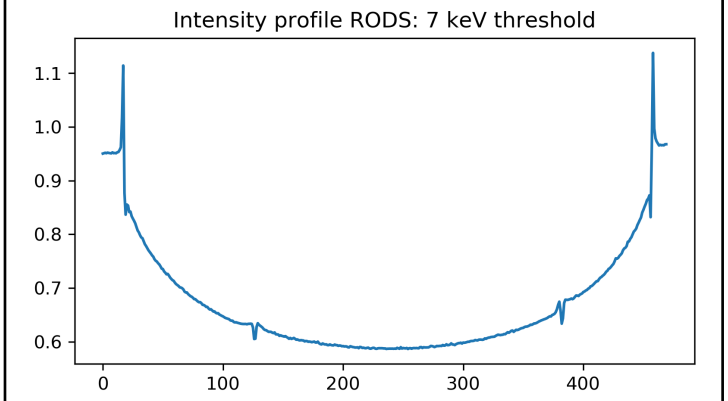
TODO:

1. ~~average the frames together to give you the best possible SNR~~
2. ~~rotate the images until they become vertical with the pixel columns~~
3. ~~subtract the two data sets to measure the difference between them to amplify subtle differences~~

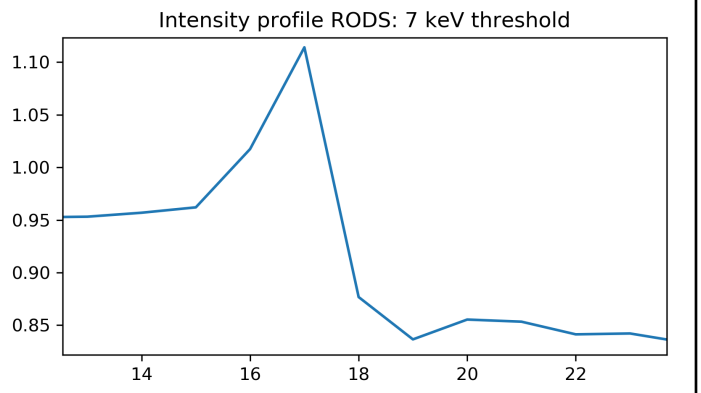
I chose to focus on the left fringe in this observations because the right fringe was somewhat increased probably due to the proximity of rod 2 and rod 4 (rod 4 ended up leaning towards rod 2 a lot)

Averaged & Rotated images SOD100cm <<<2 rods images but only 1 visualised>>>

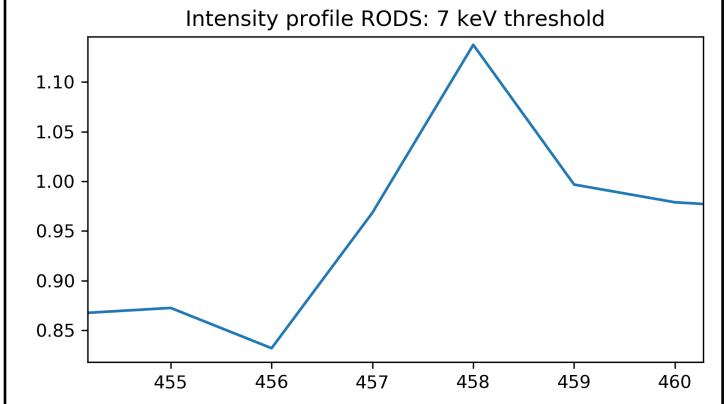




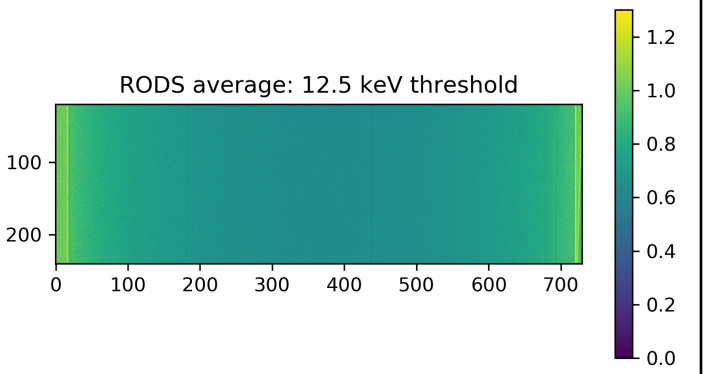
Enhanced **left fringe**

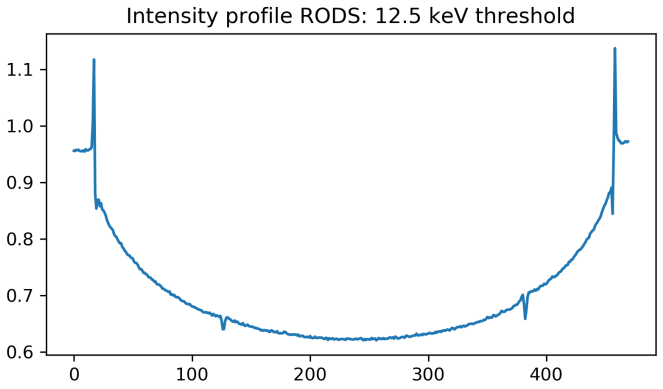


Enhanced **right fringe**



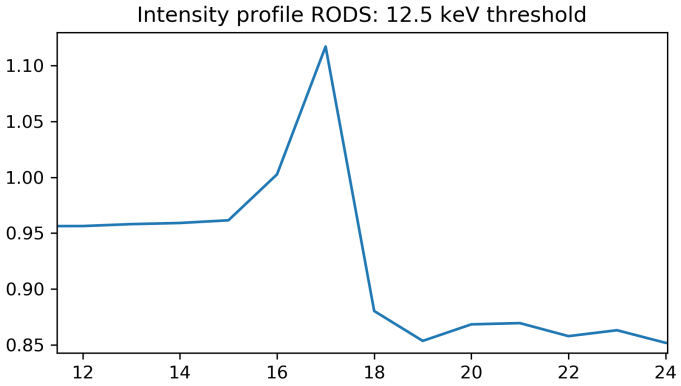
The spread looks like it is over more than 2 pixels, likely due to the second rod being too close to the first.

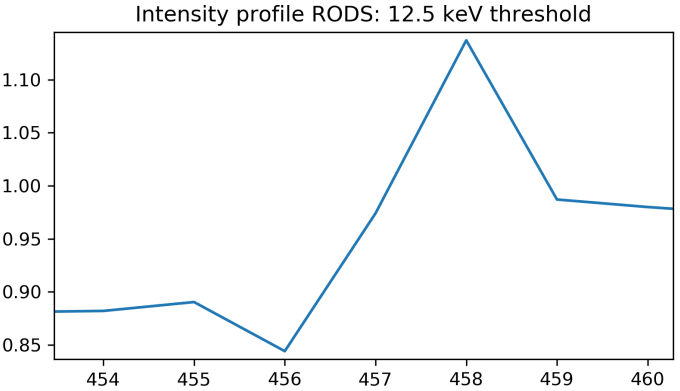


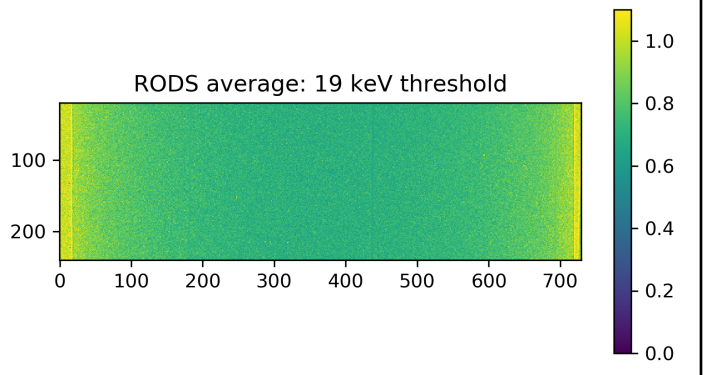


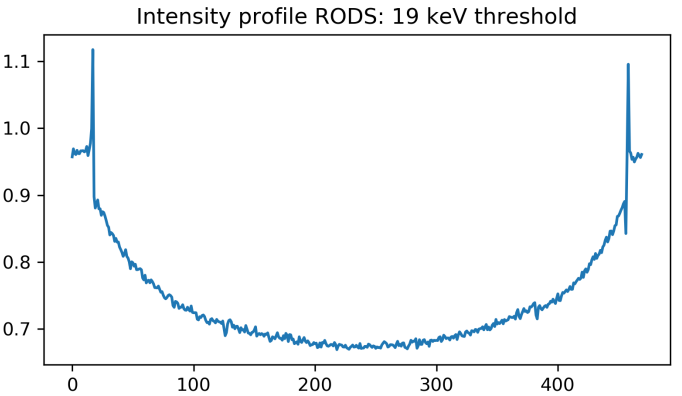
Enhanced **left fringe**

Enhanced **right fringe**



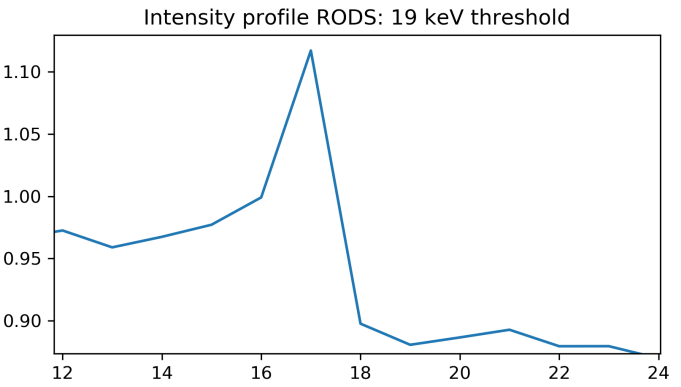


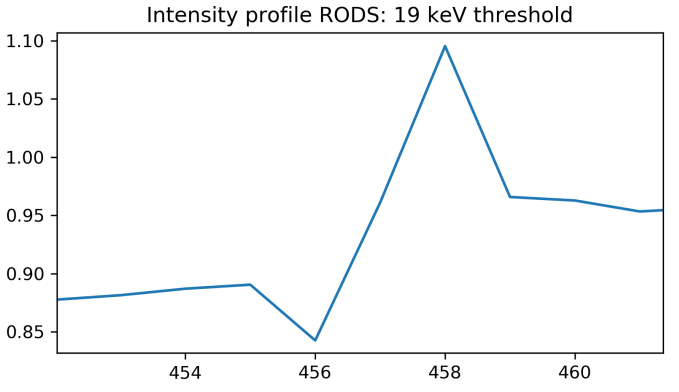




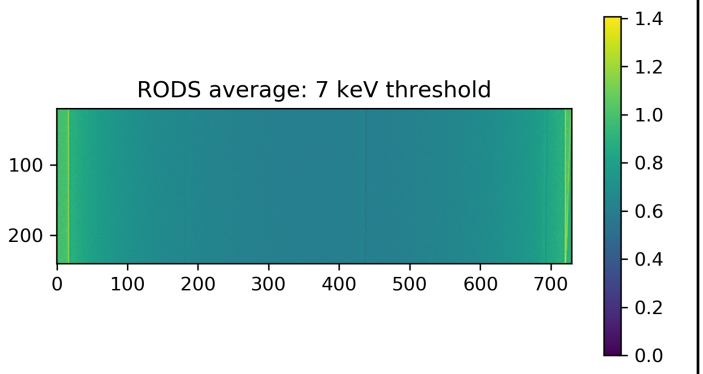
Enhanced **left fringe**

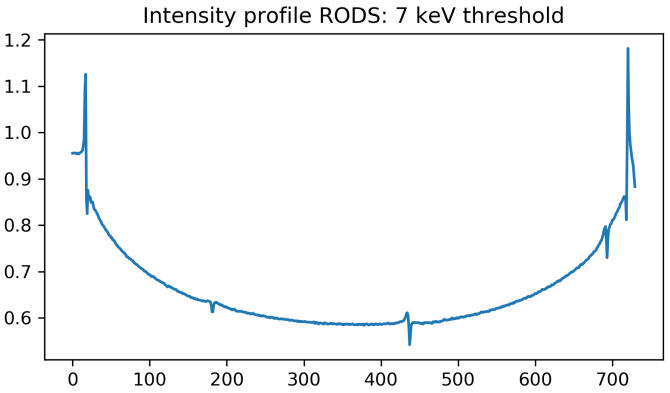
Enhanced **right fringe**





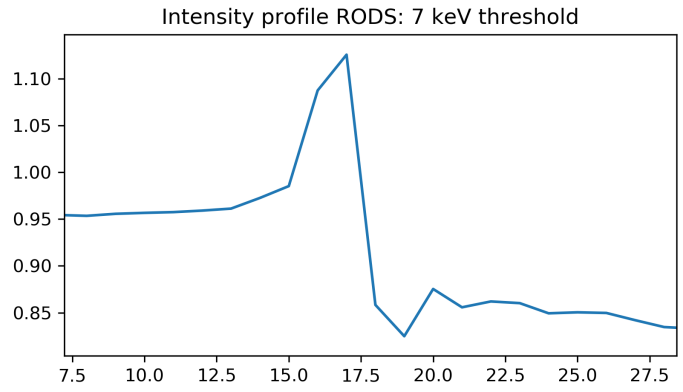
Averaged & Rotated images SOD62p5cm <<<2 rods images but only 1 visualised>>>

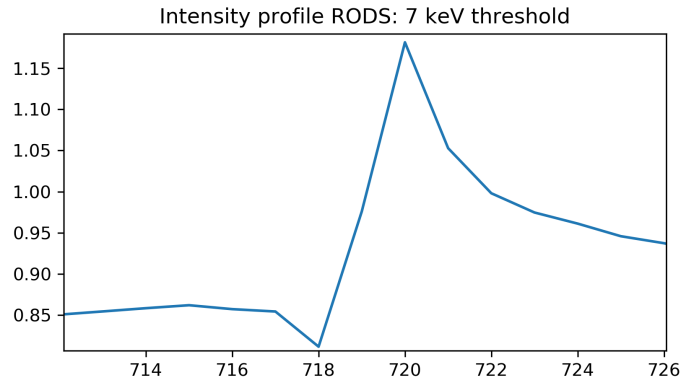


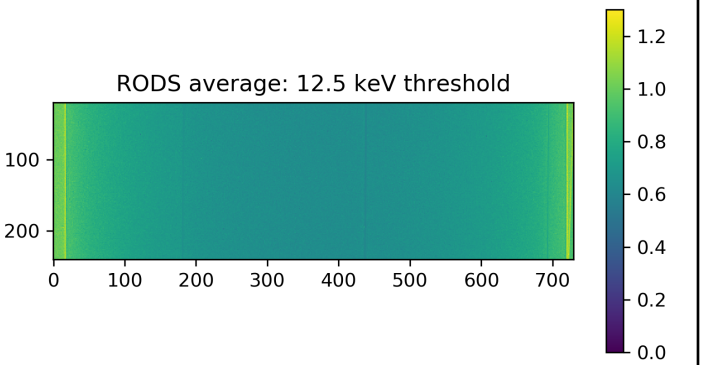


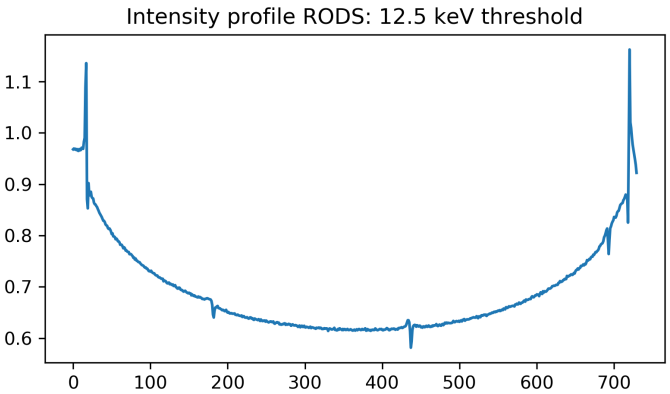
Enhanced **left fringe**

Enhanced **right fringe**



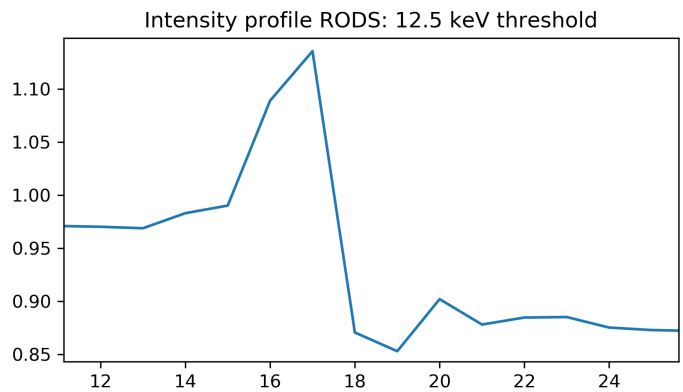


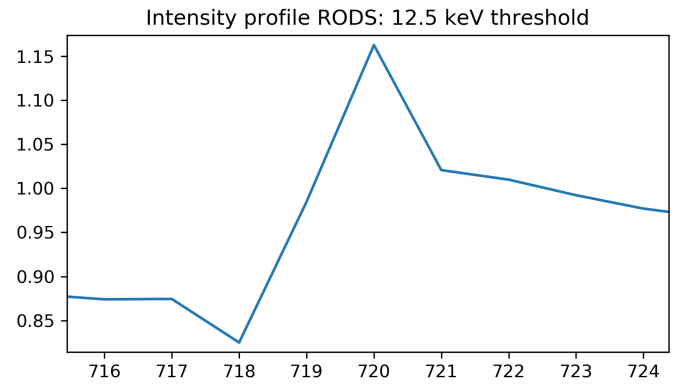


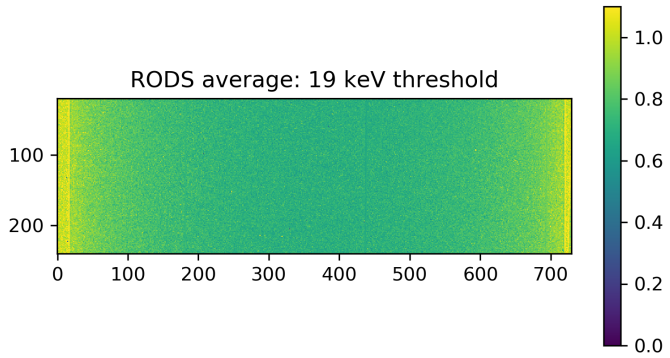


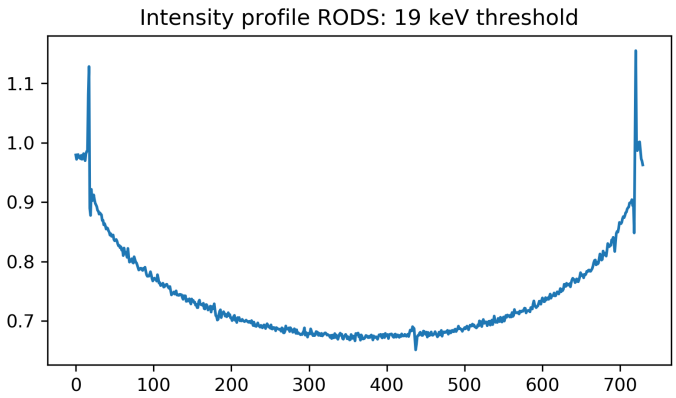
Enhanced **left fringe**

Enhanced **right fringe**



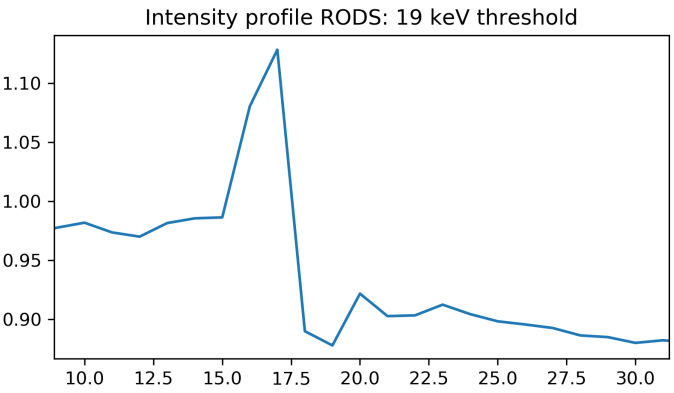


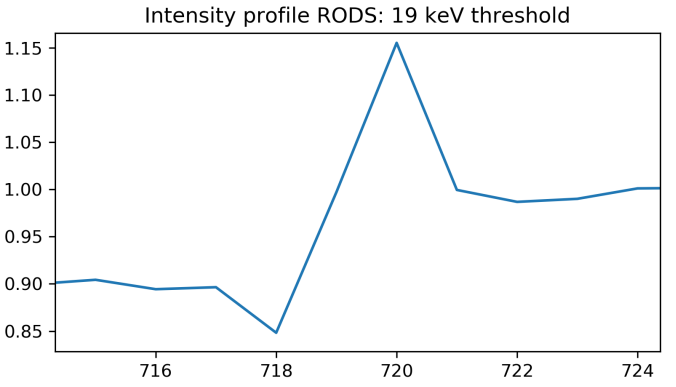




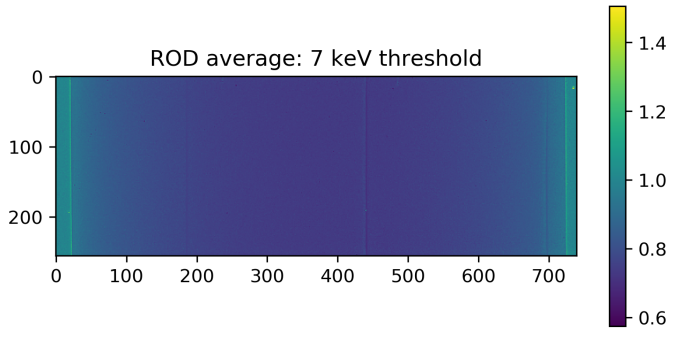
Enhanced **left fringe**

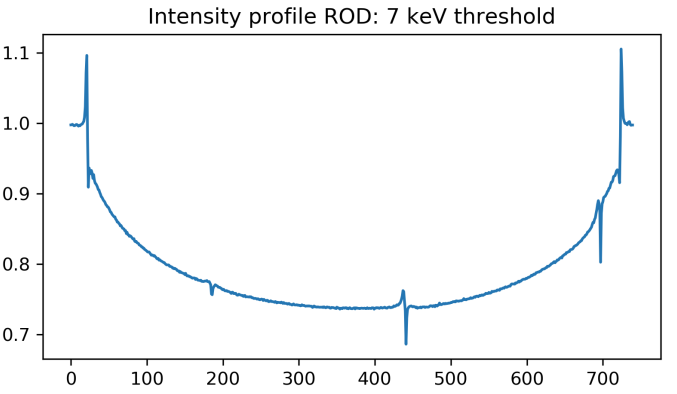
Enhanced **right fringe**





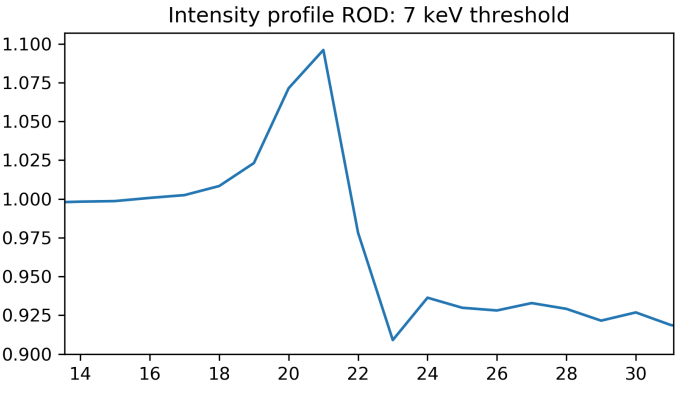
Averaged & Rotated images SOD62p5cm <<<1 rod images>>>

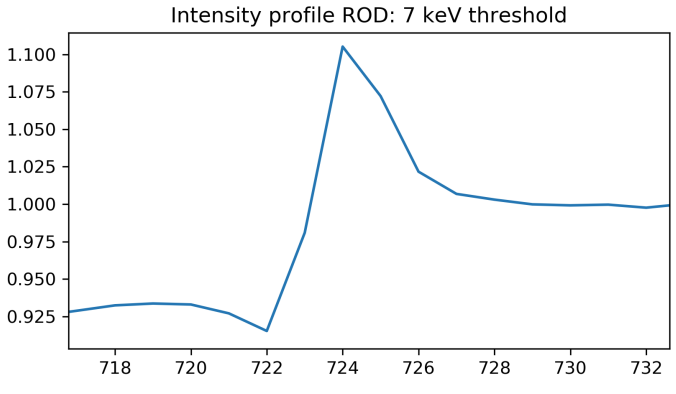


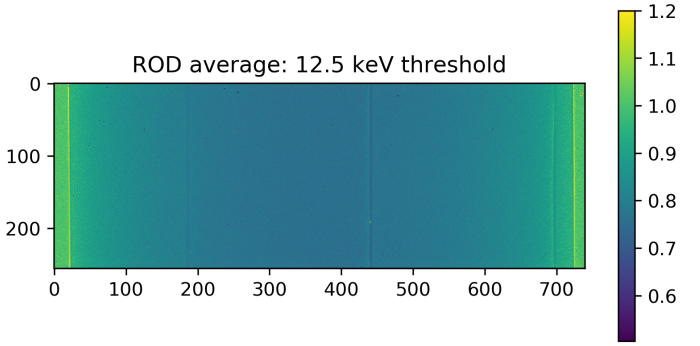


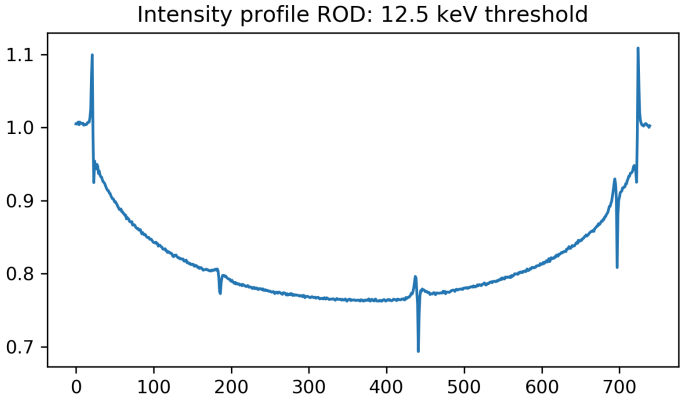
Enhanced **left fringe**

Enhanced **right fringe**



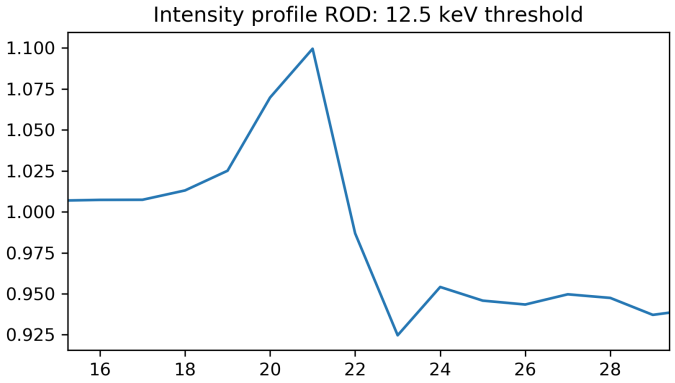


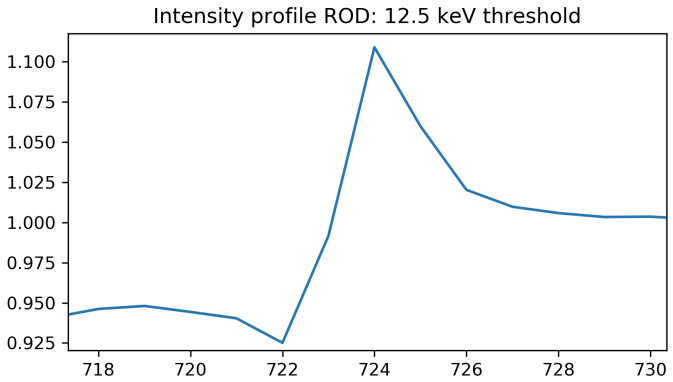


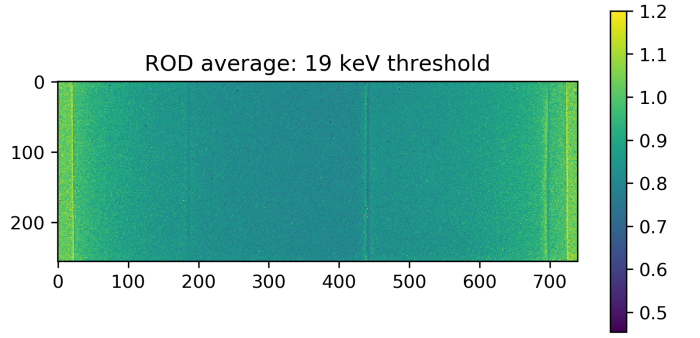


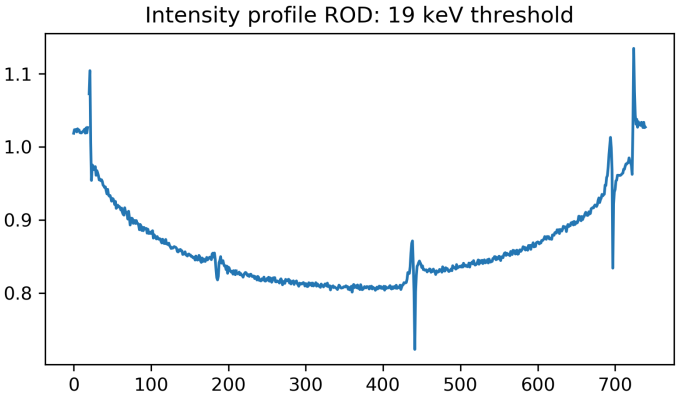
Enhanced **left fringe**

Enhanced **right fringe**



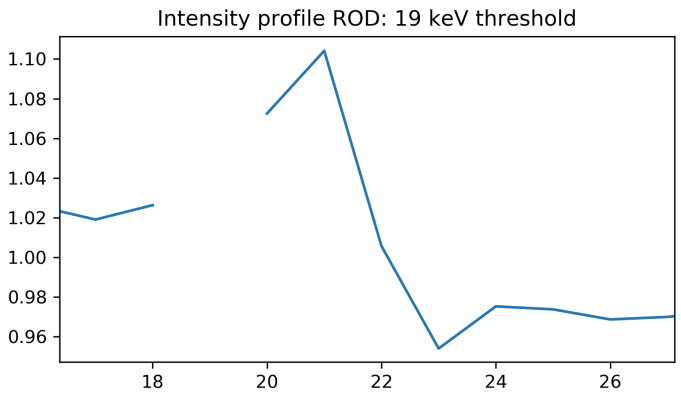




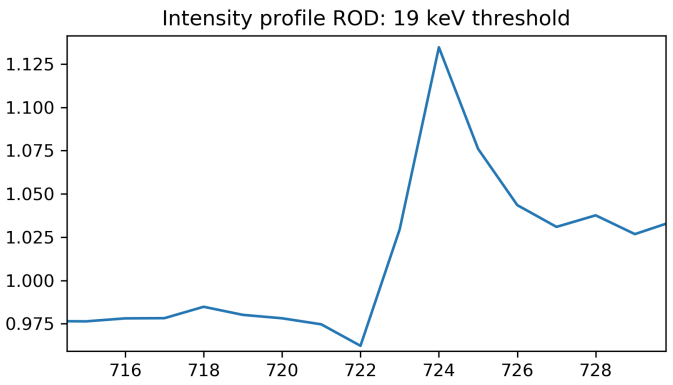


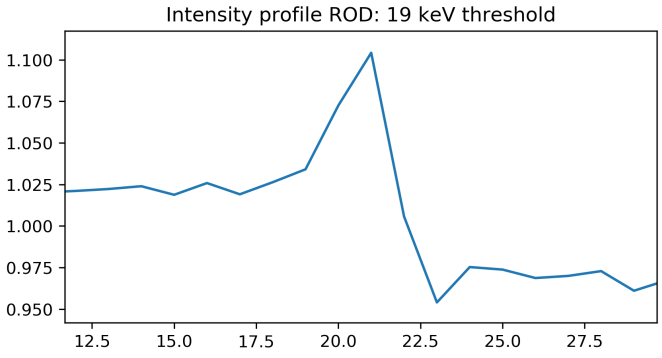
Enhanced **left fringe**

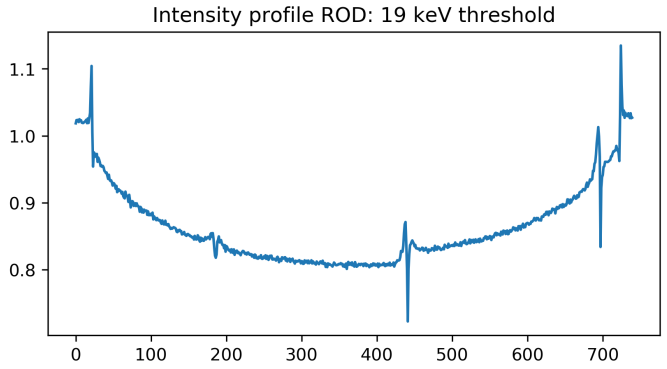
Enhanced **right fringe**



NaNs… goddamn







FIXED.

13/10/2021

MEETING DAY!

I've been double downsampling some of my simulation plots!!! (THE LAB DATA PLOTS)
This means that the IDEAL DATA plots are the correct plots to use and not the LAB DATA sim plots

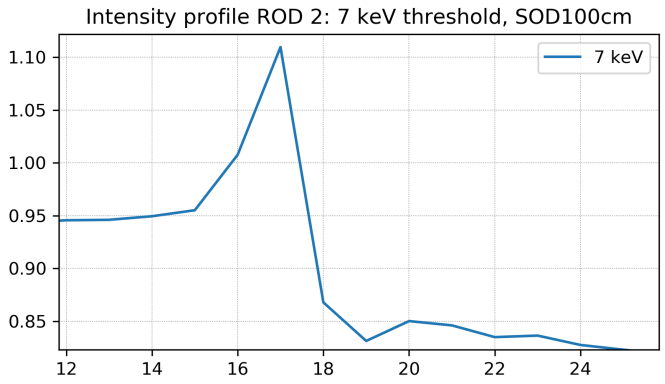
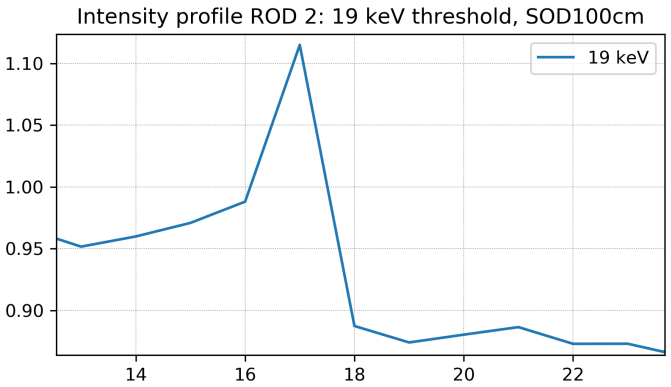
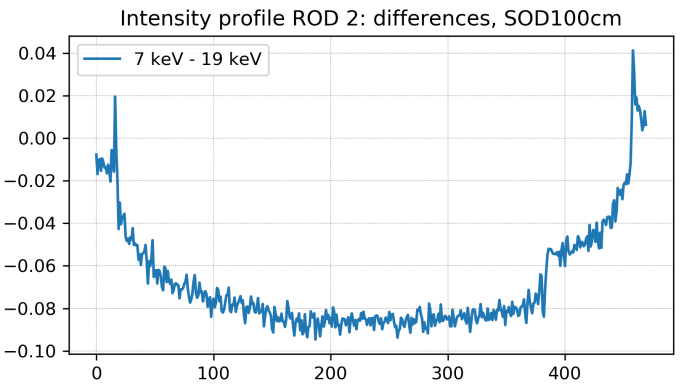
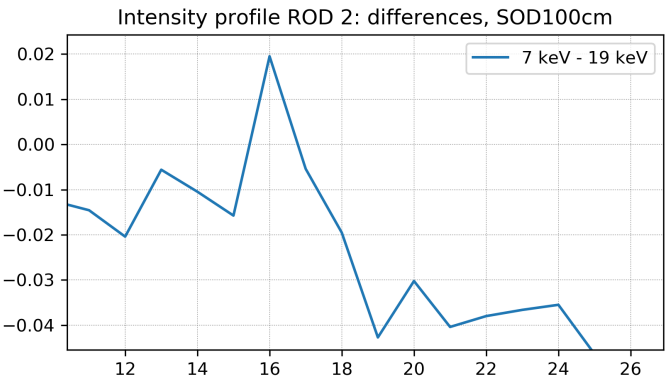
14/10/2021

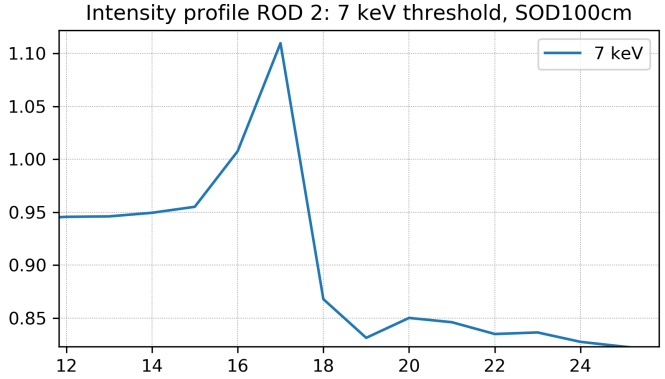
Subtract the two data sets to measure the difference between them to amplify subtle differences

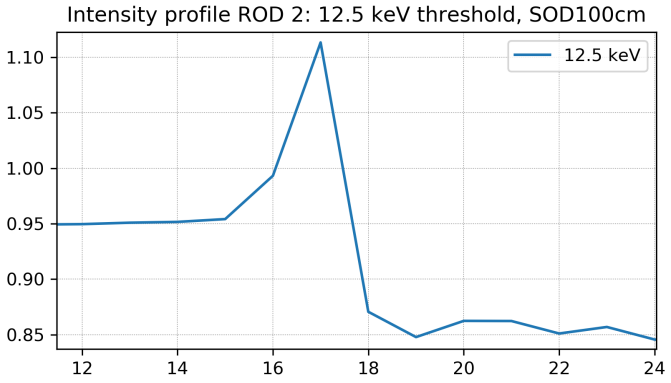
MK: *" I'd start with subtracting the 19 keV from 7 keV threshold images. These should show the greatest difference. Then I'd compare that against the 7 keV - 12.5 keV data. Hopefully you see a greater difference for the fringes when the threshold is larger."*

SOD100cm <<< only rod 2 >>> energy threshold differences

Here I am visually comparing the left fringes from the phase contrast images for each of the energy thresholds (7 keV vs 19 keV and 7 keV vs 12.5 keV) to the difference between each pair of data sets. The Tables are separated by the SOD values.
Recall that for SOD62p5 we have 2 different data sets: one comprises the images of rod 2 and rod 4, which I have cropped and rotated to make rod 2 align with the pixel columns. The second data set for SOD62p5 has only rod 2 (I have rotated this image to make the alignment of the fringes with the pixel columns better).

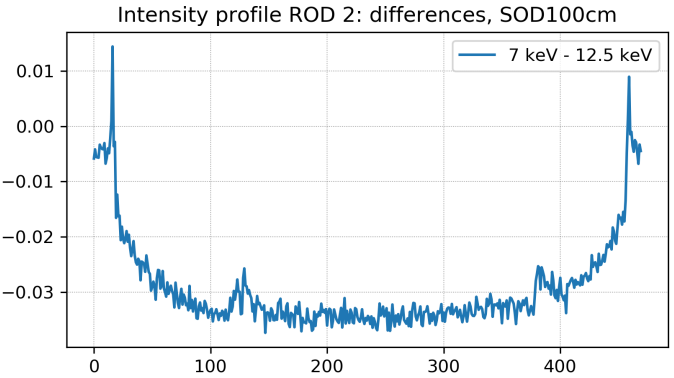
Enhanced: Left fringe (7 keV)	Enhanced: Left fringe (19 keV)
	
Difference 7 keV - 19 keV	Enhanced: Left fringe
	
Enhanced: Left fringe (7 keV)	Enhanced: Left fringe (12.5 keV)

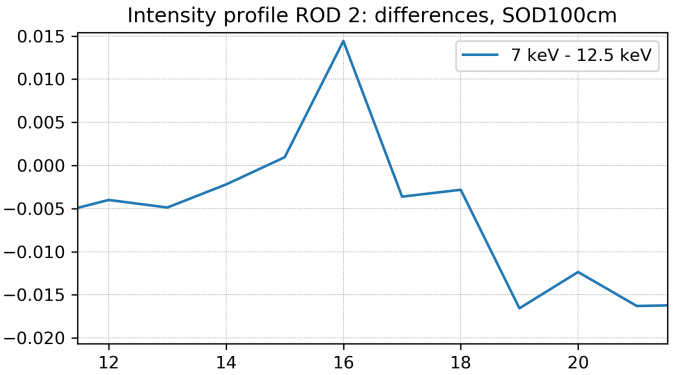




Difference 7 keV - 12.5 keV

Enhanced: Left fringe



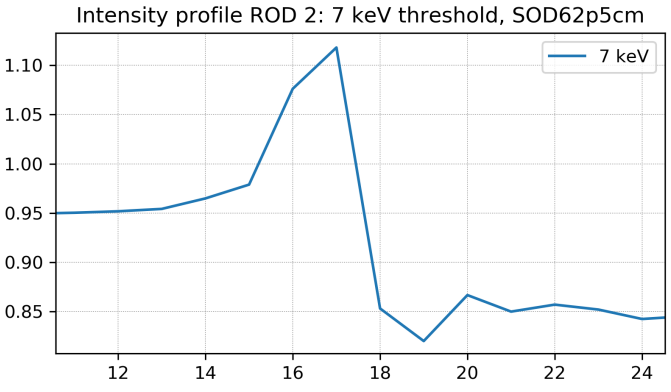


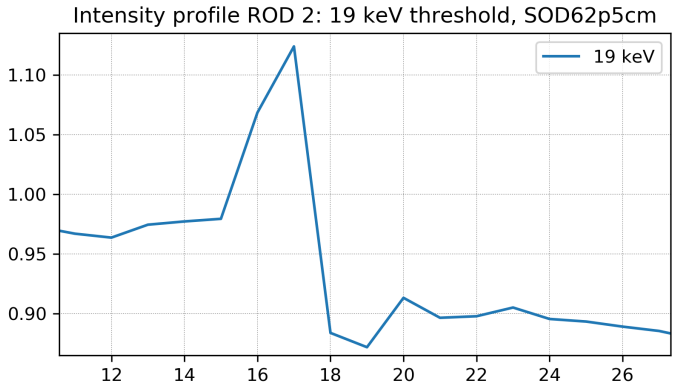
SOD62p5cm <<< only rod 2 >>> energy threshold differences

Similarly as above, I chose to focus on the left fringe in this observations because the right fringe was somewhat increased probably due to the proximity of rod 2 and rod 4 (rod 4 ended up leaning towards rod 2 a lot)

Enhanced: Left fringe (7 keV)

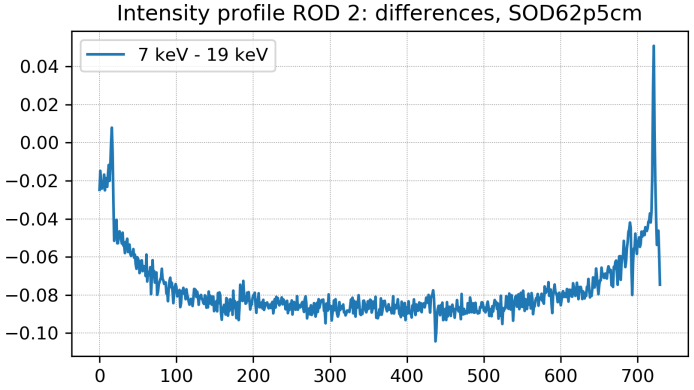
Enhanced: Left fringe (19 keV)

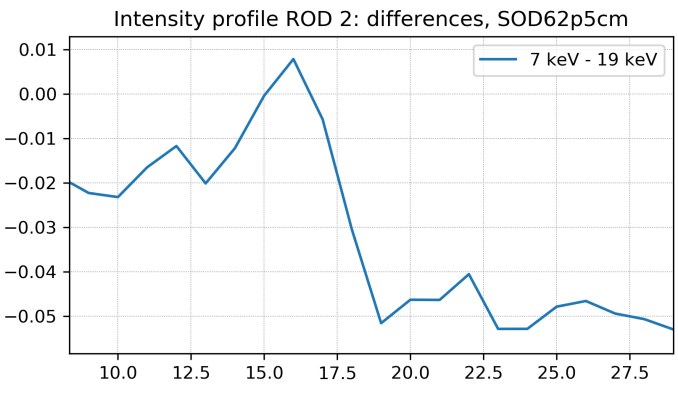




Difference 7 keV - 19 keV

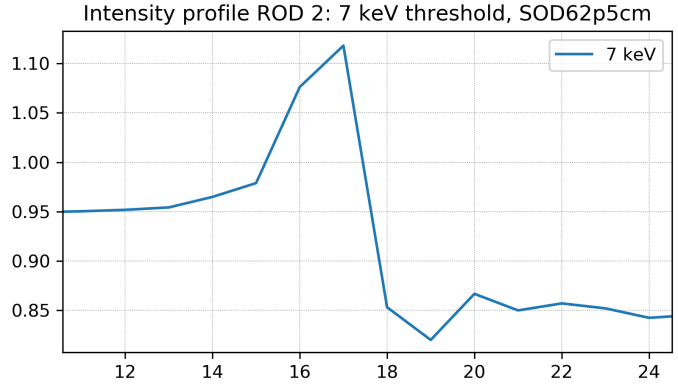
Enhanced: Left fringe

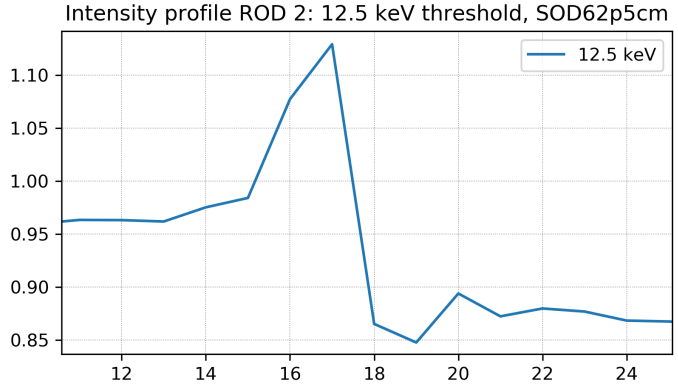




Enhanced: Left fringe (7 keV)

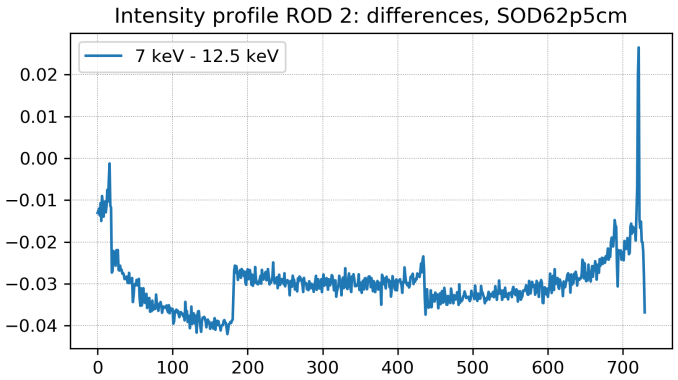
Enhanced: Left fringe (12.5 keV)

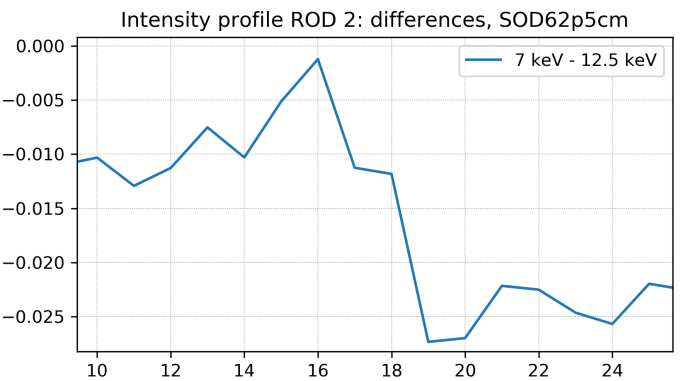




Difference 7 keV - 12.5 keV

Enhanced: Left fringe



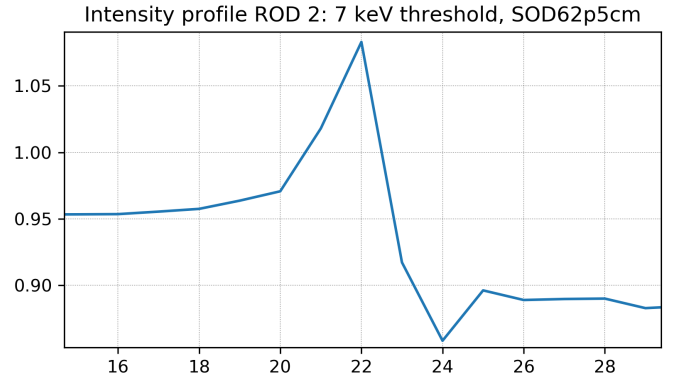


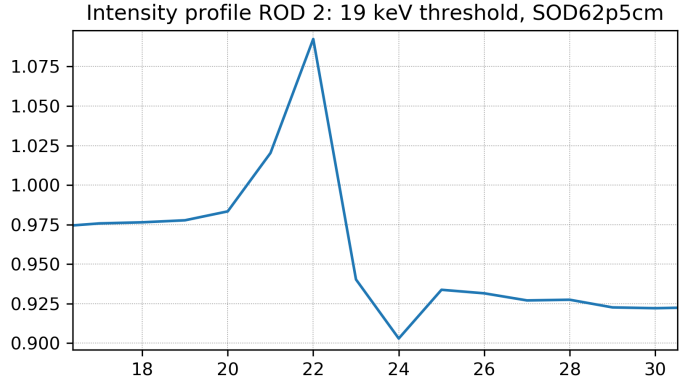
SOD62p5cm <<< rod 2 >>> energy threshold differences

This data set only comprises of a single rod (rod 2)

Enhanced: Left fringe (7 keV)

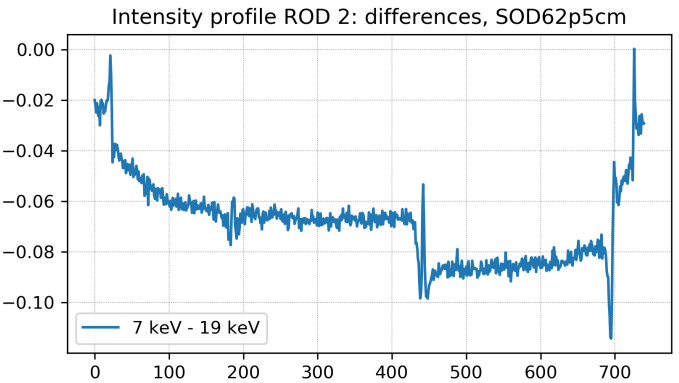
Enhanced: Left fringe (19 keV)

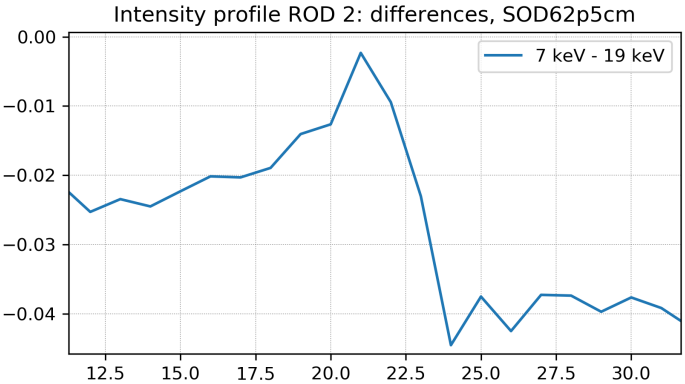




Difference 7 keV - 19 keV

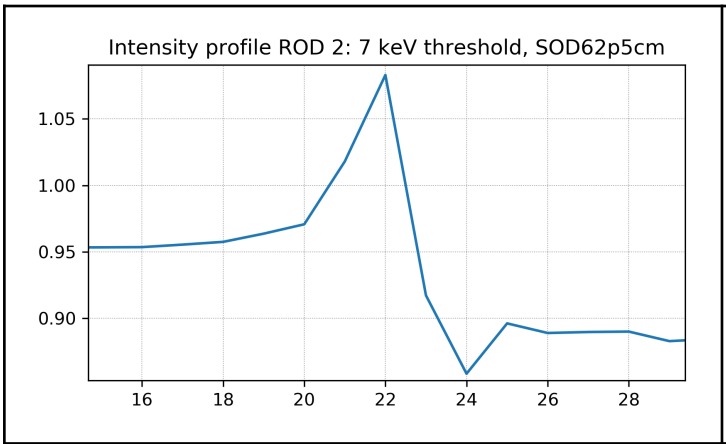
Enhanced: Left fringe

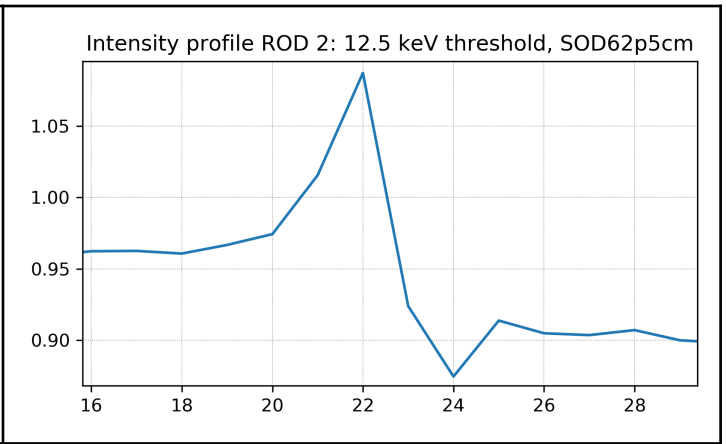




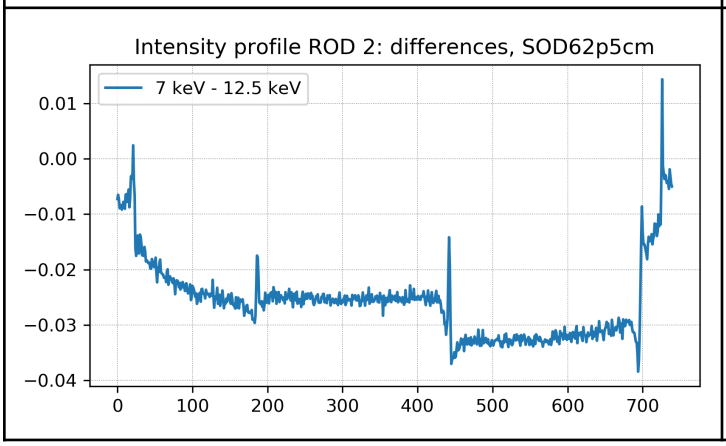
Enhanced: Left fringe (7 keV)

Enhanced: Left fringe (12.5 keV)

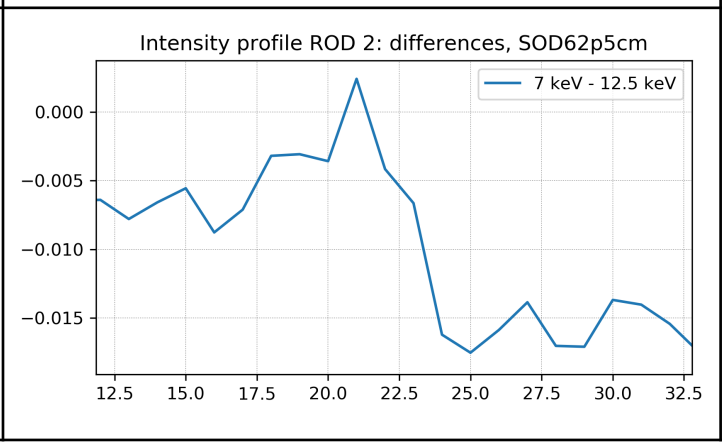




Difference 7 keV - 12.5 keV



Enhanced: Left fringe



These "fringes" that appear in some of the phase contrast images must then be the gaps between detector modules...

15/10/2021

Presentation notes

Is material density important?

BELTRAN 2011

Table 2. Values of δ and μ at 24 keV x-rays for grey/white matter and agar. These were calculated using the NIST database[7].

Material	δ ($\times 10^{-7}$)	μ (m^{-1})
Grey/white matter	4.842	56.3
Agar	3.432	40.2

For the selected energy (24 keV), ...The δ and μ values considered to be present in the agar-embedded brain sample are listed in table 2. According to the NIST database (http://www.nist.gov/index.html), grey and white matter (the main tissue types present in the brain) are **virtually identical** in reference to diagnostic energy x-ray interactions and are herein treated as identical. **We justify this statement by reference to equations (1) and (2) where we see that it is the ratio of the real and imaginary components of the refractive indices, or the difference ratio** (see equation (2)), **that is important for the Fourier filtering by the phase retrieval algorithms.** (Beltran et al. 7358)

For an object that is composed of material '1' which has complex refractive index $n = 1 - \delta_1 + i\beta_1$, an argument combining Teague's transport of intensity equation (Teague 1983) and Beer's law of attenuation implies that the projected thickness can be calculated from the measured intensity using (Paganin *et al* 2002):

$$T_1(\mathbf{r}_\perp) = -\frac{1}{\mu_1}\log_e\left(\mathbf{F}^{-1}\left\{\frac{1}{(d\delta_1/\mu_1)\mathbf{k}_\perp^2+1}\mathbf{F}\left\{\frac{I(\mathbf{r}_\perp, z=d)}{I_0}\right\}\right\}\right). \quad (1)$$

(Beltran et al. 7356)

If a medium of interest denoted by 'j' is embedded within a medium denoted as '1', their respective projected thicknesses are given by $T_j(\mathbf{r}_\perp)$ and $T_1(\mathbf{r}_\perp)$. $T_j(\mathbf{r}_\perp)$ can then be calculated using (Beltran *et al* 2010):

$$T_j(\mathbf{r}_\perp) = -\frac{1}{\mu_j-\mu_1}\log_e\left(\mathbf{F}^{-1}\left\{\frac{1}{[d(\delta_j-\delta_1)/(\mu_j-\mu_1)]\mathbf{k}_\perp^2+1}\mathbf{F}\left\{\frac{I(\mathbf{r}_\perp, z=d)}{I_0\exp[-\mu_1 A(\mathbf{r}_\perp)]}\right\}\right\}\right) \quad (2)$$

The attenuation coefficient changes as the material and its density changes because of a strong dependence on material density and/or thickness.

Moreover, variations in density of a few percent of either material

will typically have little effect on the shape of the filter in equation (2). It is therefore a valid approximation to employ a single filter for a given interface despite small density variations in inhomogeneous samples.

A tomographic reconstruction of a rat brain was made under the assumption that it comprised of a single material of variable density, which resulted in an image able to clearly distinguish between grey and white matter(Beltran et al. 7367)

$$T(x,y) = -\frac{1}{\mu}\ln\left(F^{-1}\left[\frac{F\left[I(x,y,z=\Delta)\right]/I_0}{\Delta\delta\left\|\mathbf{k}_T\right\|^2/\mu+1}\right]\right). \quad (49.29)$$ Here, F is the Fourier transform and $\mathbf{k}_T$ is the Fourier space coordinates dual to (x, y). Whilst this example may seem somewhat contrived due to the requirement of a monomorphous sample, this algorithm has proven to be extremely successful as it is highly robust against noise, unlike many alternative phase retrieval algorithms. Instability is often created by division-by-zero type artefacts. Here we see that the denominator of this Fourier space filter will never likely be zero, hence its stability. It also depends on the ratio between δ and μ, both of which are proportional to the density of the medium, hence changes in material density throughout the material are permitted. Finally, this algorithm is limited by (Pelliccia et al. 979)	If changes in material densities are allowed and δ, μ are proportional to the medium density then this projected thickness for a single changing density material should consider a ratio of δ, μ differences .

However, there is a small difference in mu between the two - otherwise we wouldn't be able to see much of any contrast in a brain CT. So we know they are small, which is why we said that in the Beltran paper and because we haven't really observed fringes. I always assumed this was because it's really just a density difference which does create absorption contrast in CT, but should not create phase contrast if density really does cancel out. I think your water/ice simulation proves there probably really are fringes, just very small ones

Works Cited

Als-Nielsen, J., and D. McMorrow. *Elements of modern X-ray physics*. John Wiley & sons Ltd., 2011.

Beltran, M. A., et al. "Interface-specific x-ray phase retrieval tomography of complex biological organs." *PHYSICS IN MEDICINE AND BIOLOGY*, vol. 56, 2011, pp. 7353–7369. https://www.flair.monash.edu.au/publications/pdfs/Beltran_etal_PhysMedBiol2011.pdf. Accessed 12 September 2021.

Beltran, M. A., et al. "2D and 3D x-ray phase retrieval of multi-material objects using a single defocus distance." *Optical Express*, vol. 18, no. 7, 2010, pp. 6423-6436. *OSA Publishing*, https://www.osapublishing.org/oe/fulltext.cfm?uri=oe-18-7-6423&id=196634. Accessed 03 08 2021.

Billington, C. J. *State-dependent forces in cold quantum gases*. 1 ed., 2018.

de la Fuente, R. "Simulating Diffraction Patterns with the Angular Spectrum Method and Python." 11 July 2021, https://rafael-fuente.github.io/simulating-diffraction-patterns-with-the-angular-spectrum-method-and-python.html#mjx-eqn%3Aeq%3A1. Accessed 06 September 2021.

Deslattes, R. D., et al. "X-ray Transition Energies (version 1.2) [Online]." *X-ray transition energies database*, National Institute of Standards and Technology, 2005, http://physics.nist.gov/XrayTrans. Accessed 21 09 2021.

Dorrer, C., and J. D. Zuegel. "Optical testing using the transport-of-intensity equation." *Optics Express*, vol. 15, no. 12, 2007, pp. 7165-7175. Accessed 03 08 2021.

Hartmann, P., et al. "Normal weight of the brain in adults in relation to age, sex, body height and weight." *Der Pathologe*, vol. 15, no. 3, 1994, pp. 165–170. *PubMed*, https://pubmed.ncbi.nlm.nih.gov/8072950/. Accessed 11 September 2021.

Hasgall, P. A., et al. "IT'IS Database for thermal and electromagnetic parameters of biological tissues." 15 May 2018, https://itis.swiss/virtual-population/tissue-properties/database/density/. Accessed 14 September 2021.

Mackenzie, R. J. *Gray Matter vs White Matter*. content piece. 20 August 2019. *Technology Networks, Neuroscience news & research*, Technology Networks, https://www.technologynetworks.com/neuroscience/articles/gray-matter-vs-white-matter-322973. Accessed 11 September 2021.

NIST. "X-Ray Form Factor, Attenuation, and Scattering Tables." *NIST Standard Reference Database 66*, 21 September 2009, https://physics.nist.gov/PhysRefData/FFast/html/form.html. Accessed 11 September 2021.

The NumPy community. "numpy.fft.fftfreq." *NumPy Documentation*, 2021, https://numpy.org/doc/stable/reference/generated/numpy.fft.fftfreq.html. Accessed 09 08 2021.

Paganin, D. *Coherent X-Ray Optics*. 1 ed., Oxford University Press, 2006. Accessed 25 August 2021.

Paganin, D., et al. “Simultaneous phase and amplitude extraction from a single defocused image of a homogeneous object.” *Journal of Microscopy*, vol. 206, no. 1, 2002, pp. 33-40. *Wiley Online Library*, https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-2818.2002.01010.x. Accessed 02 October 2021.

Paganin, D., and D. Pelliccia. “Tutorials on X-ray Phase Contrast Imaging: Some Fundamentals and Some Conjectures on Future Developments.” 2019. *arXiv*, https://arxiv.org/abs/1902.00364v2. Accessed 13 08 2021.

Pelliccia, D., et al. *Handbook of X-Ray Imaging: Physics and Technology*. 1 ed., vol. 47, Boca Raton, FL, CRC Press Taylor & Francis Group, 2018.

Phelps, M. E., et al. “Attenuation Coefficients of Various Body Tissues, Fluids, and Lesions at Photon Energies of 18 to 136 keV.” *Radiology*, vol. 117, no. 3, 1975, pp. 573-583. *Radiological Society of North America*, https://pubs.rsna.org/doi/10.1148/117.3.573. Accessed 12 September 2021.

Roper, L. D. *Using Sigmoid and Double-Sigmoid Functions for Earth-States Transitions. http://roperld.com/science/*, http://roperld.com/science/DoubleSigmoid.pdf.

Schulz, G., et al. “High-resolution tomographic imaging of a human cerebellum: comparison of absorption and grating-based phase contrast.” *Interface*, vol. 7, 2010, pp. 1665-1676. *Sci Hub*, https://sci-hub.se/https://doi.org/10.1098/rsif.2010.0281. Accessed 21 September 2021.

Summerfield, M. *Rapid GUI programming with Python and Qt the definitive guide to PyQt programming*. 6 ed., Upper Saddle River; NJ, Prentice Hall, 2012.

Wikipedia, the free encyclopedia. “Compton scattering.” 10 July 2021, https://en.wikipedia.org/wiki/Compton_scattering. Accessed 23 July 2021.

Wikipedia, the free encyclopedia. “Phase-contrast imaging.” *Phase-contrast imaging*, 2021, https://en.wikipedia.org/wiki/Phase-contrast_imaging. Accessed 19 July 2021.

Zuo, C., et al. “Transport of intensity equation: a tutorial.” *Optics and Lasers in Engineering*, vol. 135, 2020, p. 106187. *ScienceDirect - Elsevier*, https://www.sciencedirect.com/science/article/pii/S0143816619320858. Accessed 21 07 2021.