PHS3350 Logbook

Ana Fabela Hinojosa ID: 27876594

"What I cannot create I do not understand" -R. Feynman

Week -∞ (25/01/21 – 01/03/21)

I started reading the two text resources suggested by Jesper and Meera (J&M):

- ➔ Non-Hermitian Quantum Mechanics by Nimrod Moiseyev
- ➔ Making sense of non-Hermitian Hamiltonians by Carl M Bender

Aims:

- Get through the first chapter of N Moiseyev "Non-Hermitian quantum mechanics".
- Familiarise myself with the motivations behind the non-Hermitian formalism.

25/01/21
Since my goals are so open at this stage; I think a good way to start to understand the topic is to read the first chapter of Moiseyev "Non-Hermitian quantum mechanics". (NHQM) as thoroughly as possible and attempt each exercise. There are four exercises in this chapter.

Notes on chapter 1.1 of Moiseyev:

Different formulations of quantum mechanics

The first part of the chapter revisits the standard formalism of QM and makes emphasis on the association of real valued eigenvalues of operators representing observable quantities. The claim is that QM requires that the operators used to represent observables satisfy a criterion called Hermicity.
"an operator is **Hermitian** if it operates on functions belonging to Hilbert space ($\mathfrak{H}$) of square integrable functions such that if f, g are square integrable, or have asymptotes which are periodic functions, they satisfy: $\langle f|H|g\rangle = \langle g|H|f\rangle^*$." (Moiseyev 3)

28/01
What I understand so far is that apparently Hermicity is not as fundamentally necessary as it has been thought so far. Going ahead and allowing this idea to stick around regardless of the lack of a deeper understanding of Moiseyev's ideas, it appears that using the NHQM formalism we can study more types of systems!

The standard Hermitian formalism cannot easily (or sometimes at all) explain certain phenomena such as self orthogonality or coalesced degenerate **resonance** states.
I don't really understand what "resonance states" means at this point. 1
Moiseyev explains that NHQM is useful when system dynamics can be described by some number of resonance states. (?)

Exercise 1.1 From (Moiseyev 4)

Exercise 1.1

Since the standard formulation of quantum mechanics defines physical operators as Hermitian we have to stress here that **the Hermitian property of an operator is heavily dependent on the basis set which is used to represent a dynamical variable by matrices of infinite order**. It is commonly assumed that these matrices obey the usual laws valid for finite matrices. However, it is obvious that this is not necessarily always true. Show that $\hat{p}_x^3 = (-\mathrm{id}/\mathrm{d}x)^3 \equiv \mathrm{id}_x^3$ is not Hermitian when particle-in-a-box eigenfunctions, $\{\phi_n(x)\}_{n=1,2,\ldots}$, are used as a basis set. Associate it with the fact that $\mathbf{PP}^2 \neq \mathbf{P}^2\mathbf{P}$, where $\mathbf{P}$ is an infinite order Hermitian matrix that represents the momentum operator for a particle in a box, and its square $\mathbf{P}^2$ is well defined and diagonal.

I have no clue of what I am supposed to do.
While googling about Moiseyev I encountered a series of lectures given by him at Technion Israel Institute of Technology this is the playlist link: Non Hermitian Quantum Mechanics

03/02/21
I asked Chris about this problem and he suggested that I check the following equality:

$$(P_x^3)_{nm} \stackrel{?}{=} (P_x^3)^*_{mn}$$

To do this, I have to represent the momentum operator as a matrix, transpose it and conjugate it to verify the equality status of the generic matrix elements. First I must find the states for a 1D particle-in-a-box.

08/02/21

(1) 1D particle-in-a-box eigenfunctions are found by

- we require continuity of function & 1st derivative
- wavefunction is zero in the infinite potential region
  $\phi_n(0) = 0$ and $\phi_n(a) = 0$

we solve $H\phi_n = E\phi_n$

with $H = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}$ inside the box

let $\phi_n(x) = 0$ outside the box

∵ constant coeff 2nd order evolution becomes: $\frac{-\hbar^2}{2m}\lambda^2 = E$

$$\therefore \lambda^2 = -\frac{2mE}{\hbar^2} \Rightarrow \lambda = \sqrt{\frac{2mE}{\hbar^2}}\, i$$

$$\phi_n(x) = A\cos\left(\sqrt{\frac{2mE}{\hbar^2}}\, x\right) + B\sin\left(\sqrt{\frac{2mE}{\hbar^2}}\, x\right)$$

To satisfy BCS: when $x=0$ $\phi_n(0) = A = 0$

when $x=L$ $\phi_n(a) = B\sin\left(\sqrt{\frac{2mE}{\hbar}}\, L\right) = 0$

we need $\sqrt{\frac{2mE}{\hbar}} = n\pi$

if $x=L \quad \phi_n(L) = B\sin\left(\frac{\sqrt{2mE}}{\hbar}L\right) = 0$

we need $\frac{\sqrt{2mE}}{\hbar}L = n\pi$ which means that $E = \frac{n^2\pi^2\hbar^2}{2mL^2}$

using Normalisation to find the constant B

$$\langle \phi_n(x) \mid \phi_n(x) \rangle = |B^2| \int_0^L \left|\sin\left(\tfrac{\sqrt{2mE}}{\hbar}x\right)\right|^2 dx = |B^2| \int_0^L \left|\sin\left(\tfrac{n\pi x}{L}\right)\right|^2 dx = 1$$

let $u = \frac{n\pi x}{L} \Rightarrow \frac{du}{dx} = \frac{n\pi}{L} \Rightarrow \frac{L\,du}{n\pi} = dx$ if $x=L \Rightarrow u = n\pi$, $x=0 \Rightarrow u=0$

$$|B^2| \int_0^{n\pi} \left|\sin\left(\tfrac{n\pi x}{L}\right)\right|^2 dx = |B^2| \frac{L}{n\pi} \int_0^{n\pi} \sin^2(u)\,du$$

$\because \sin^2(u) = \frac{1-\cos(2u)}{2}$

$$\therefore \frac{|B^2|L}{2n\pi} \int_0^{n\pi} 1-\cos(2u)\,du = \frac{|B^2|L}{2n\pi}\left(u + \tfrac{1}{2}\sin(2u)\right)_0^{n\pi} = 1$$

$$\therefore \frac{|B^2|L}{2n\pi}\left(u + \tfrac{1}{2}\sin(2u)\right)_0^{n\pi} = 1$$

$\therefore B^2 = \frac{2}{L}$ Then: $\phi_n(x) = \sqrt{\frac{2}{L}}\sin\left(\frac{\sqrt{2mE_n}}{\hbar}x\right)$

where $E_n = \frac{n^2\pi^2\hbar^2}{2mL^2}$

$$(P_x^3)_{nm} \stackrel{?}{=} (P_x^3)^*_{mn}$$

we find the matrix elements by evaluating:

$$(P_x^3)_{nm} = \langle \phi_n | \hat{P}_x^3 | \phi_m \rangle = \langle \phi_n | \left(-i\tfrac{d}{dx}\right)^3 | \phi_m \rangle$$

$$= \sqrt{\tfrac{2}{L}}\sin\left(\tfrac{\sqrt{2mE_n}}{\hbar}x\right)\left(-i\tfrac{d}{dx}\right)^3 \sqrt{\tfrac{2}{L}}\sin\left(\tfrac{\sqrt{2mE_m}}{\hbar}x\right)$$

$$= -i\tfrac{2}{L}\sin\left(\tfrac{\sqrt{2mE_n}}{\hbar}x\right)\left(\tfrac{d}{dx}\right)^3 \sin\left(\tfrac{\sqrt{2mE_m}}{\hbar}x\right) \quad \text{let } \tfrac{\sqrt{2mE_m}}{\hbar} = s \quad \therefore \tfrac{d^3}{dx^3}\sin(sx) = -s^3\cos(sx)$$

$$= -i\tfrac{2}{L}\sin\left(\tfrac{\sqrt{2mE_n}}{\hbar}x\right)\left(-\left(\tfrac{\sqrt{2mE_m}}{\hbar}\right)^3 \cos\left(\tfrac{\sqrt{2mE_m}}{\hbar}x\right)\right)$$

$$= -i\tfrac{2}{L}\left(\tfrac{\sqrt{2mE_m}}{\hbar}\right)^3 \sin\left(\tfrac{\sqrt{2mE_n}}{\hbar}x\right)\cos\left(\tfrac{\sqrt{2mE_m}}{\hbar}x\right)$$

$$(P_x^3)^*_{mn} = \langle \phi_m | \hat{P}_x^3 | \phi_n \rangle^* = \langle \phi_m | \left(-i\frac{d}{dx}\right)^3 | \phi_n \rangle^*$$

$$= \left(\sqrt{\frac{2}{L}} \sin\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \left(-i\frac{d}{dx}\right)^3 \sqrt{\frac{2}{L}} \sin\left(\sqrt{\frac{2mE_n}{\hbar}}x\right)\right)^*$$

$$= \left(-i\frac{2}{L} \sin\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \left(\frac{d}{dx}\right)^3 \sin\left(\sqrt{\frac{2mE_n}{\hbar}}x\right)\right)^*$$

$$= \left(-i\frac{2}{L} \sin\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \left(-\left(\sqrt{\frac{2mE_n}{\hbar}}\right)^3 \cos\left(\sqrt{\frac{2mE_n}{\hbar}}x\right)\right)\right)^*$$

$$= i\frac{2}{L}\left(\sqrt{\frac{2mE_n}{\hbar}}\right)^3 \sin\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \cos\left(\sqrt{\frac{2mE_n}{\hbar}}x\right)$$

$$(P_x^3)_{nm} \stackrel{?}{=} (P_x^3)^*_{mn}$$

$$-i\frac{2}{L}\left(\sqrt{\frac{2mE_m}{\hbar}}\right)^3 \sin\left(\sqrt{\frac{2mE_n}{\hbar}}x\right) \cos\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \neq i\frac{2}{L}\left(\sqrt{\frac{2mE_n}{\hbar}}\right)^3 \sin\left(\sqrt{\frac{2mE_m}{\hbar}}x\right) \cos\left(\sqrt{\frac{2mE_n}{\hbar}}x\right)$$

The matrix is non-Hermitian!

10/02/21
Notes on chapter 1.2 and 1.3 of Moiseyev:

Sources of non Hermicity

1. *Potential terms supporting a continuous spectrum*
   - Naturally some systems have Non-Hermitian Hamiltonians (NHH). Such Hamiltonians describe open systems such as systems undergoing decay and dissipative processes.
     In this section there is a more concise but still general description of so-called **resonance states**. Moiseyev describes resonance states as Metastable states: Metastable resonance states are such where systems have enough energy to break into many non-interacting subsystems but they don't necessarily do so.
   - Metastable resonance states can be manufactured.
   - "It is very natural to associate the metastable resonance states of the system with stationary solutions of the TISE with outgoing asymptotes rather than with non- stationary wave packet solutions of the TDSE"(Moiseyev 5)
     I think this means that we can use the TISE even in time-dependent resonance problems.
   - Resonance states appear in the non Hermitian sector of the domain of Hamiltonian therefore they seem to be well defined in NHQM (associated with a single eigenstate of the Hamiltonian).

28/02/21

2. *Complex local potentials*

   Complex perturbation potential terms Introduced to the Hamiltonian.
   Reflection-free absorbing potentials (RF-CAP) are used to avoid no-physical interference effects from reflections on the boundary given the quantisation of space method used in numerical solutions to DEs. (Interactions between a propagating wave packet and the boundary should not actually take place.)

   In this section of the chapter Moiseyev mentions NHH which are not related to resonance phenomena, but are NH from the inclusion of purely imaginary external fields. For example"
   $V = igx^3$ where $g \in \mathbb{R}$ and the Hamiltonian commutes with:

   The $\mathscr{PT}$ symmetry operator

   The $\mathscr{PT}$ symmetry operator:
   $\mathscr{P}$ is the parity (space inversion) operator
   $\mathscr{P}x\mathscr{P}$ = -x and $\mathscr{P}p\mathscr{P}$ = -p
   $\mathcal{P}^2 = \mathbb{1}$
   This operator is linear

   $\mathscr{T}$ is the time reversal (complex inversion) operator
   $\mathscr{T}x\mathscr{T}$ = x, $\mathscr{T}p\mathscr{T}$ = -p and $\mathscr{T}i\mathscr{T}$ = -i
   $\mathcal{T}^2 = \mathbb{1}$
   This operator is antilinear

   Hamiltonian H is $\mathscr{PT}$ symmetric if $H^{\mathcal{PT}} = (\mathcal{PT})H(\mathcal{PT}) = H$

Exercise 1.2 From (Moiseyev 5)

Exercise 1.2

Often in experiments the detectors measure the momenta of the outgoing particles/subsystems. Because of the uncertainty relation in quantum mechanics we know that it is impossible to measure precisely both the positions and the momenta of the outgoing particles/subsystems. However, it is possible to build detectors (antennas) that measure precisely the momenta of the outgoing particles/subsystems. Is there violation of the uncertainty "principle" since we know precisely the location of the antenna that measures the momenta of the outgoing particles/subsystems?

No, since we are not measuring the position of the system. (we only measure the momentum of each outgoing subsystem)

Note: The detailed methodology for obtaining the solutions as they appear in figure 1.1 and 1.2 in Moiseyev is not very clear to me. They are supposed to be explained more thoroughly in subsequent chapters.

Exercise 1.3 From (Moiseyev 10)

Exercise 1.3

Show by using semiclassical arguments that the spectrum of non-Hermitian Hamiltonian $\hat{H} = \hat{p}^2 + igx^3$ is real and positive for any value of $g \neq 0$ provided the solution eigenfunctions are defined on the whole real line.

finding the energy spectrum $\Leftrightarrow$ solving eigenvalue problems

Then I need to make a matrix?

and calculate: $\det(H - \lambda I) = 0$ to find eigenvalues?

rewrite this hamiltonian in $\hat{H}\psi = E\psi$

$\therefore (\hat{p}^2 + ig\hat{x}^3)\psi = E\psi$

$\therefore \left(-\frac{d^2}{dx^2} + igx^3 - E\right)\psi = 0$

? ? ?

--- in which basis?

ALSO:

"provided the solution eigenfunctions are defined in the whole real line"

This doesn't seem trivial.

01/03/21

Important points:

- NHQM aims to describe time dependent phenomena whilst using the time independent Schrödinger equation.
- There are different origins for non-Hermitian operators.
- NHH can have a real energy spectrum.
- $\mathscr{PT}$ symmetry operator.
- Don't panic

Week 1 (01/03/21 – 07/03/21)

Aims:

1. Solve Exercise 1.3 in Moiseyev
2. Prepare for meeting (week 2) (Show J&M what I have learned so far)

Tasks:

1. ~~Meeting with J&M (Wednesday 14:30)~~
   Organise project details (what do I actually need to do?)
2. ~~Attend group meeting~~
3. ~~Start reading: Understanding non-Hermitian Hamiltonians by CM Bender~~

01/03/21
Notes on Making Sense of non-Hermitian Hamiltonians by CM Bender

Some of the fundamental axioms of quantum mechanics

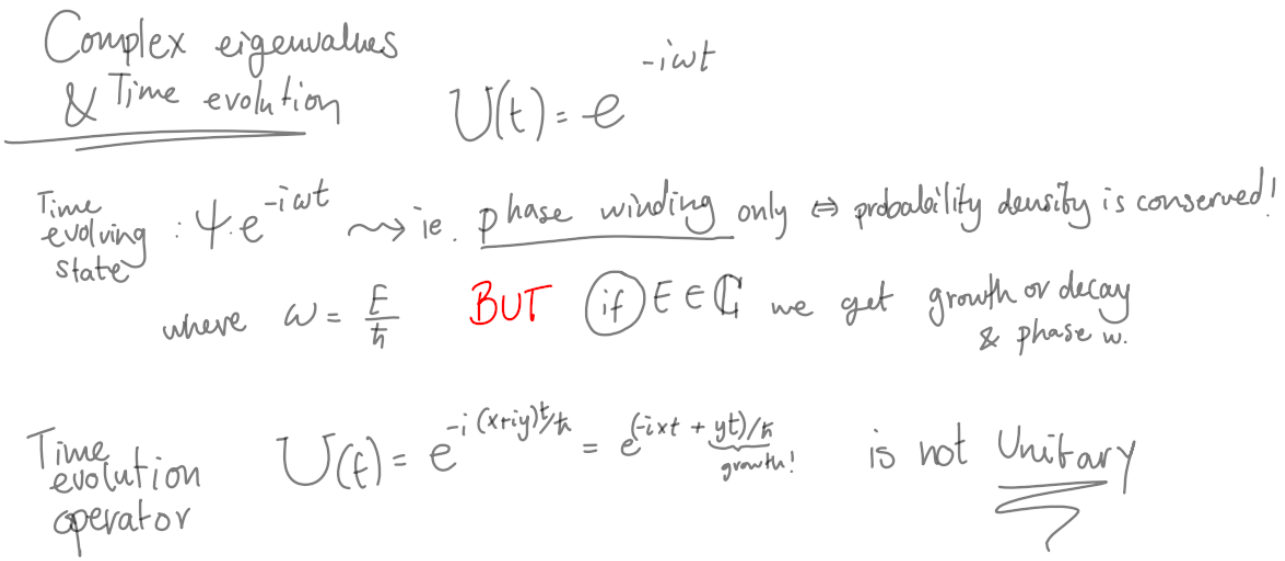
Hamiltonian properties:

- Real valued and Bounded below energy spectrum
- Unitary time evolution
  Does this axiom imply that QM must always study closed systems? What about dissipative processes? YEP… that's why Hemicity is a postulate.
- Lorentz covariance and causality
  I don't know what this is, I will investigate it
- Hermicity

Why are non-Hermitian Hamiltonians not more widely used?

Bender explains that G Barton claims in his 1963 Introduction to Advanced Field Theory (Bender 950)

- NHHs do not have a unitary S-matrix.
  An S-matrix relates the initial and final states of physical systems undergoing scattering
  Therefore if we don't have unitary evolution then there is no conservation of probability.



- NHH have complex eigenvalues
  But according to NHQM this is not always true!
  Some NHH have a subset of real valued eigenvalues.

$\mathcal{PT}$- symmetry as a substitute for Dirac Hermicity

IF a Hamiltonian is $\mathcal{PT}$- symmetric ie. $H = H^{\mathcal{PT}} = (\mathcal{PT})H(\mathcal{PT})$ then It satisfies $[(\mathcal{PT}), H] = 0$

03/03/21
MEETING Day!
I had my first meeting of the semester with J&M. We spoke about the tasks I need to fulfil while doing my research. At the moment I don't think I have a very specific task other than getting familiar enough with NHQM to be able to give them a little talk next week about my current understanding on the topic, They suggested I check out one more resource, a paper by Elena Ostrosvskaya's group. **Observation of non-Hermitian degeneracies in a chaotic exciton-polariton billiard**. I think that the goal here for me is to be to eventually understand the spatially dependent 2x2 Hamiltonian which is non-Hermitian.

04/03/21
(Bender 955)

Determining the eigenvalues of a $\mathcal{PT}$-symmetric Hamiltonian

The $\mathcal{PT}$operator is non-linear. It's eigenstates may or may not be shared with those of the Hamiltonian.
If we assume ψ is an eigenstate of both H and $\mathcal{PT}$(Bender 955)

$\mathcal{PT}\psi = \lambda\psi$ (1)
multiply by $\mathcal{PT}$ on the left and use $(\mathcal{PT})^2 = \mathbb{1}$: (WHY?)
$\therefore \psi = \mathcal{PT}\lambda(\mathcal{PT})^2\psi$

[2] since $\mathcal{T}i\mathcal{T} = -i$:
$\therefore \psi = \lambda^*\lambda\psi$ ⇝ I don't understand this step.
· where does the extra λ come from?
· I understand that $\mathcal{PT}\lambda = \lambda^*\mathcal{P}$

04/03/21
I am wrong! · $\mathcal{PT}\lambda \neq \lambda^*\mathcal{P}$
Antilinear mapping (conjugate linear)
$f: V \to W$
$f(ax+by) = \bar{a}f(x) + \bar{b}f(y)$
$\forall a,b \in \mathbb{C}$
but in fact $\mathcal{PT}\lambda\psi = \lambda^*\mathcal{PT}\psi$
$\therefore \mathcal{PT}\psi = \lambda\psi$
$\therefore \psi = \mathcal{PT}\lambda\psi$
antilinearity of $\mathcal{T}$:
$\therefore \psi = \lambda^*\mathcal{PT}\psi = \lambda^*\lambda\psi$

The extra λ comes from a simple substitution

05/03/21
So, here we are:
$\psi = \lambda^*\lambda\psi = |\lambda|^2\psi$
$\therefore |\lambda|^2 = 1$
$\therefore \lambda = e^{i\phi}$ for some $\phi \in \mathbb{R}$
using equation (1) & $(\mathcal{PT})H\psi = (\mathcal{PT})E(\mathcal{PT})^2\psi$ and recalling: $H(\mathcal{PT}) - (\mathcal{PT})H = 0$
$\Rightarrow H\lambda\psi = (\mathcal{PT})E(\mathcal{PT})\lambda\psi$
$\therefore E\lambda\psi = E^*\lambda\psi$ $\quad \because (\mathcal{PT})E(\mathcal{PT}) = E^*$
Then since $\lambda \neq 0 \Rightarrow E = E^*$
ALTHOUGH THIS IS GENERALLY FALSE!

Definition:
If every eigenfunction of a $\mathcal{PT}$- symmetric Hamiltonian is also an eigenfunction of the $\mathcal{PT}$operator then we say that the $\mathcal{PT}$- symmetry of the Hamiltonian is unbroken, otherwise the symmetry is broken (Bender 956)

Week 2 (08/03/21 – 14/03/21)

Aims:

1. Reproduce Fig 1 in Bender
2. Simulate a RF-CAP
   (Similar to the one presented in video 5 in Moiseyev/Technion youtube series)
   01-5 - Numerical needs for non hermitian Hamiltonian

I need to check if aims are adequate by J&M standards.

Tasks:

1. ~~Meeting with J&M (Wednesday 14:30)~~
2. ~~Attend group meeting~~

08/03/21

Using this definition and the method I learned above I can try to solve exercise 1.3 of Moiseyev:

(1) showing that the spectrum of non-Hermitian Hamiltonian $H = \hat{p}^2 + ig\hat{x}^3$ is real and positive for any value of $g \neq 0$

(2) provided the solution eigenfunctions are defined in the whole real line.

for (1): we can show that if $\hat{H} = \hat{p}^2 + ig\hat{x}^3$ has unbroken $\mathcal{PT}$-symmetry then it's energy spectrum is real. (but not necessarily positive)

• Showing that $\hat{H}$ is $\mathcal{PT}$-symmetric (unbroken) •

IF $\hat{H} = (\mathcal{PT})\hat{H}(\mathcal{PT})$ Then $\hat{H}$ is invariant under Space-time reversal!

$$\therefore (\mathcal{PT})\hat{H}(\mathcal{PT}) = (\mathcal{PT})(\hat{p}^2 + ig\hat{x}^3)(\mathcal{PT})$$
$$= (\mathcal{PT})(\mathcal{P})(\hat{p}^2 - ig\hat{x}^3)(\mathcal{T})$$
$$= (\mathcal{PT})(\mathcal{PT})(\hat{p}^2 + ig\hat{x}^3)$$
$$\therefore (\mathcal{PT})\hat{H}(\mathcal{PT}) = (\mathcal{PT})^2\hat{H}$$
$$\therefore (\mathcal{PT})\hat{H}(\mathcal{PT}) = \hat{H}$$

which means $\hat{H}$ is $\mathcal{PT}$ symmetric ✳

{ NOTE: Parity inversion doesn't work if power is even
{ ie. $\mathcal{P}\hat{p}^2\mathcal{P} = \hat{p}^2$

To determine the spectrum

Since $\hat{H}$ and $\mathcal{PT}$ share eigenstates ✳

using the eigenvalue equation

$$\hat{H}\psi = E\psi \qquad (2)$$

multiply by $\mathcal{PT}$, and insert identity operator

$$(\mathcal{PT})\hat{H}\psi = (\mathcal{PT})E(\mathcal{PT})^2\psi$$

since $\hat{H} = \hat{H}^{\mathcal{PT}}$ ⇔ $\mathcal{PT}$ and $\hat{H}$ commute!

$$\hat{H}(\mathcal{PT})\psi = (\mathcal{PT})E(\mathcal{PT})^2\psi$$

By eqn (1):

$$\hat{H}\lambda\psi = E^*(\mathcal{PT})\psi$$
$$\therefore \hat{H}\lambda\psi = E^*\lambda\psi$$
$$\therefore E\lambda\psi = E^*\lambda\psi$$
$$\therefore E = E^* \quad \therefore E \in \mathbb{R}$$

{ How can it be exclusively positive?

{ also.. ψ are required to be defined in the WHOLE $\mathbb{R}$-line how is this relevant?

Note: This method is NOT semiclassical

I should in addition try to solve the eigenvalue problem as a DE. as per the solution in [0], ie. The Bohr-Sommerfeld Quantisation rule.

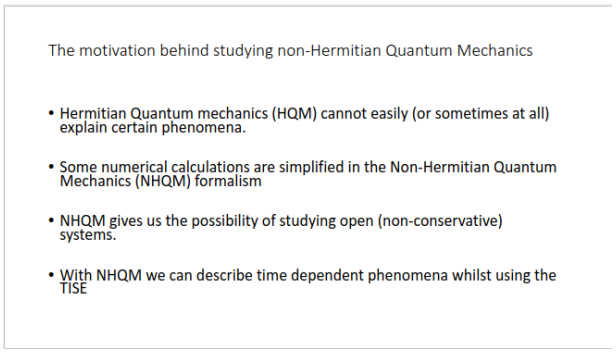
∴ We can prove that a $\mathcal{PT}$ symmetric Hamiltonian has a real energy spectrum if we prove it's $\mathcal{PT}$ symmetry is unbroken.

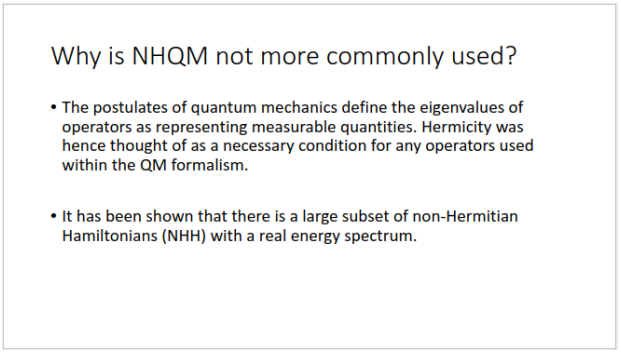
In both Moiseyev and Bender there are examples of what could be consider the same type of NHH:

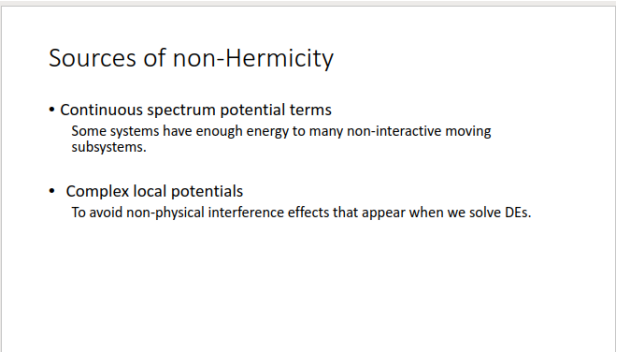
In ex 1.3 of Moiseyev the Hamiltonian is $H = p^2 + igx^3$ whilst in Bender $H = p^2 + x^2(ix^\varepsilon)$ where $g, \varepsilon \in \mathbb{R}$, $g \neq 0$. Clearly both parameters are important But Bender emphasises ε.

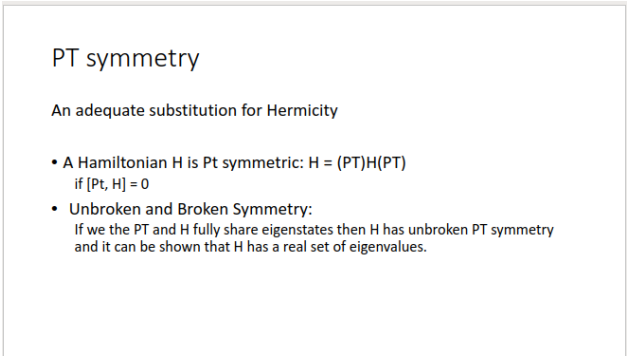
In the caption of Fig 1 in (Bender 952) it is explained that when $\varepsilon \geq 0$ the spectrum is real and positive and the energy rises with ε. The behaviour when $-1 < \varepsilon < 0$ is of finite (but decreasing number of) real eigenvalues but infinite complex conjugate pairs. For $\varepsilon \leq -1$ the ground state energy diverges and there are no real eigenvalues left.

10/03/21
MEETING Day!
I made some slides to present to J&M to have a guide over what I wanted to say.
These are the slides:









J&M seemed enthused about the progress so far.
They were happy with my suggestion of having Aims 1 and 2 as my goals for the time being.
They also suggested I add this to my aims

3. Try to understand E Gao et al.'s (2x2 Hamiltonian)
   Compare Gao et al. to Bender page 986 to see if comparison is useful.

I will start reading this paper this week and for the time being drop the idea of solving as many of moiseyev's problems as I can.

13/03/21
Notes from (Moiseyev 2)
"The numerical propagation of wave packets is much more simple when taken within the framework
of the non-Hermitian formalism of quantum mechanics rather than in the standard (Hermitian) formalism. **This is due to the inclusion of a reflection-free complex absorbing potential (RF-CAP) in the Hamiltonian which attains non-zero values only in the non-interacting region in the coordinate space where the physical potentials vanish.**
This approach enables one to avoid the artificial reflections from the edge of the numerical grid when a finite number of grid points (or a finite number of basis functions) are used to describe a propagated wave-packet."

Making an RF-CAP simulation

Requires I construct a wave function with analytically verifiable derivatives.
I'll do a propagating gaussian with IC $\psi(x,0) = e^{-(x^2/2\sigma^2)}e^{ikx}$
I need:

- an array of x values, k values, and a σthat resolves the gaussian given the array of x.
- The RF-CAP (probably I will try V = igx^3given that is an example in Moiseyev)
- The 1D time dependent Schrödinger equation
- 4th order Runge Kutta for time evolution

TODO:

1. ~~Update physunits package~~
2. ~~Make git repo (RF-CAP)~~
3. ~~Read fftfreq documentation~~

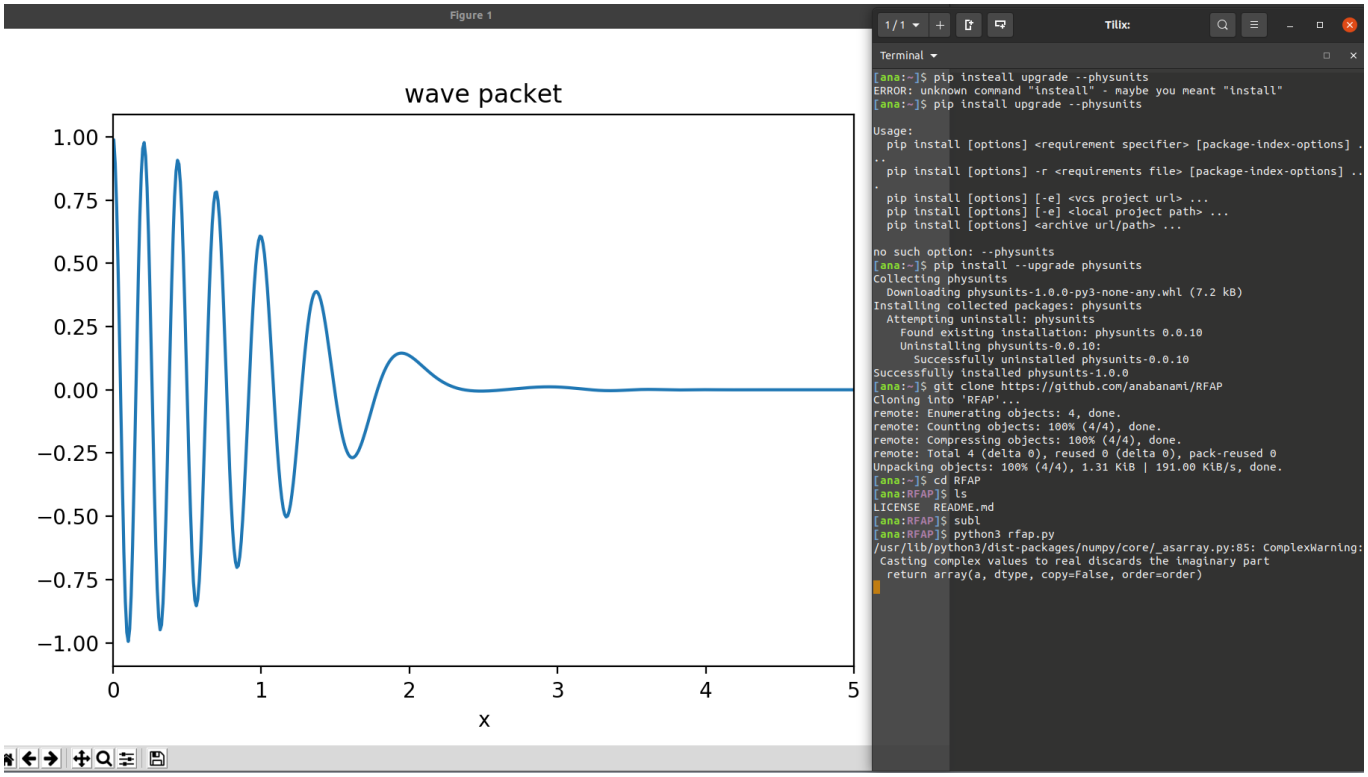
First: verify I can plot the stationary wave packet

~/RFAP/rfap.py • - Sublime Text

File Edit Selection Find View Goto Tools Project Preferences Help

sayings | untitled | rfap.py

```
# PHS3350
# Week 2 - wave packet and RFAP -
# Ana Fabela Hinojosa, 13/03/2021

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import physunits
plt.rcParams['figure.dpi'] = 200

σ = 1
x_max = 5
x = np.linspace(-x_max,x_max, 1024)
n = x.size
x_step = 0.1
k = 2 * np.pi * np.fft.fftfreq(n, x_step)


wave = np.exp(- x**2 / (2*σ**2)) * np.exp(1j*k*x)


plt.plot(x, wave)
plt.xlim(0,5)
plt.xlabel("x")
plt.title('wave packet')

plt.show()
```

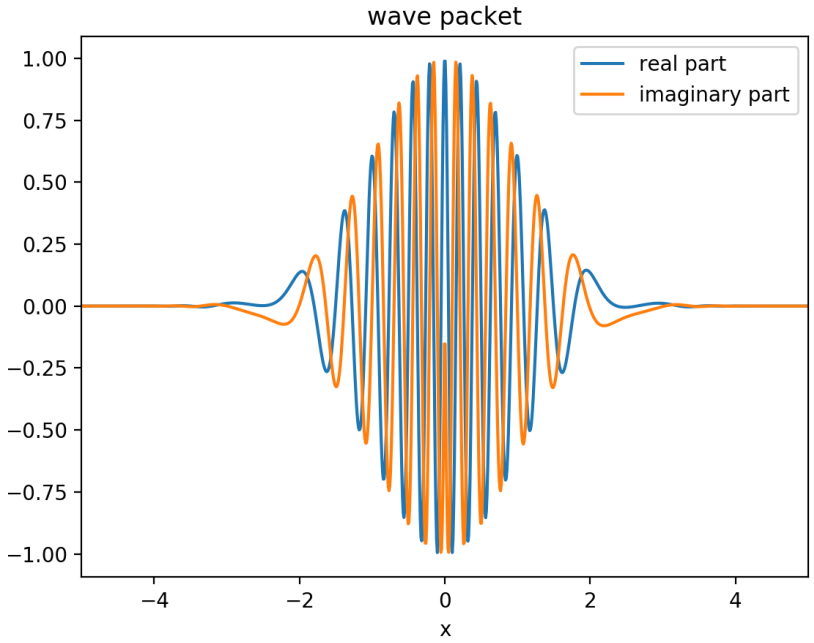


I get this ERROR:

/usr/lib/python3/dist-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part

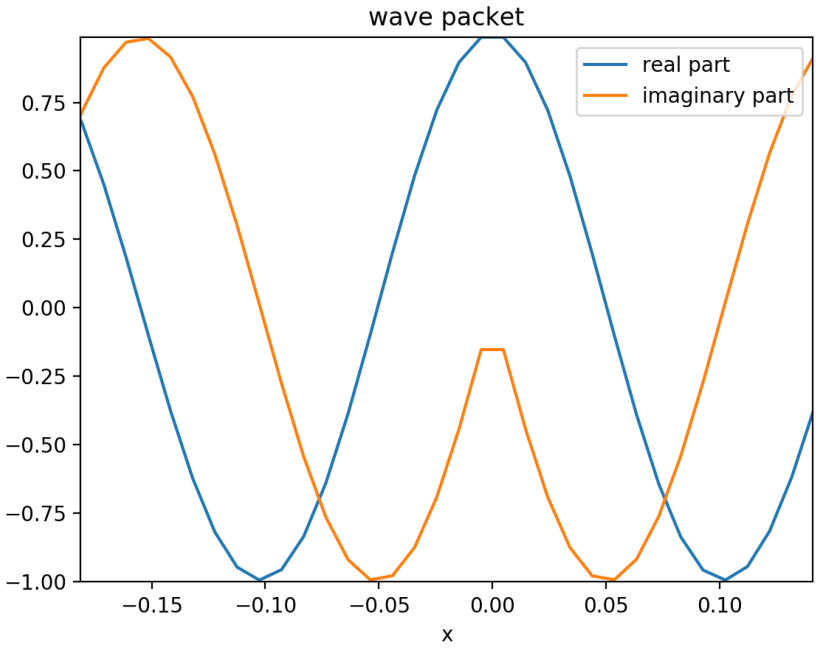
return array(a, dtype, copy=False, order=order)

I want to see both the imaginary and real parts of the wave, therefore I googled the error and found a fix



It's better but the imaginary part looks funny near the origin.

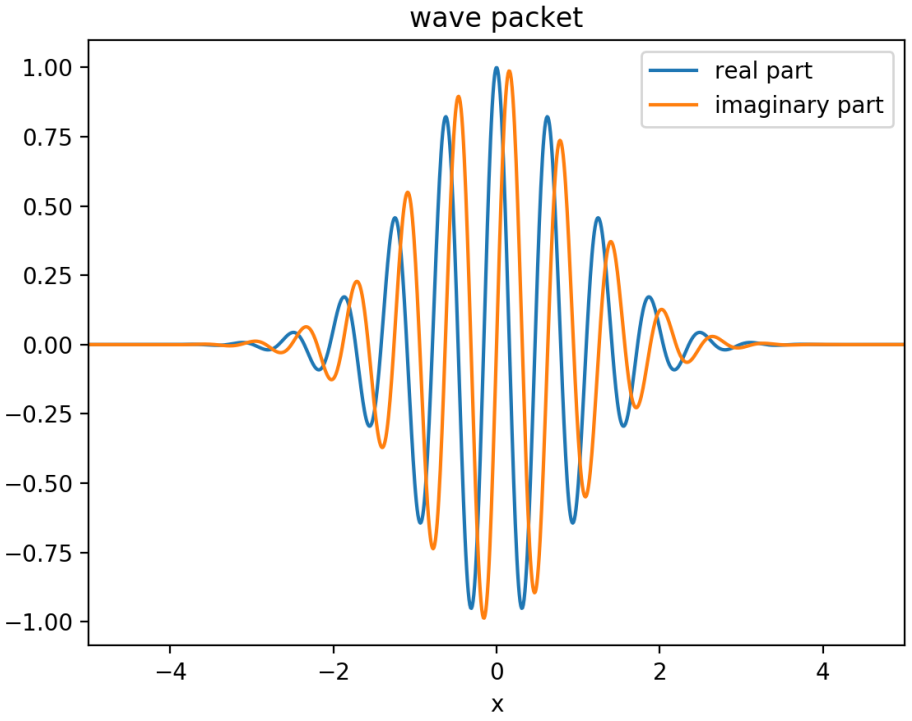
This is what it looks like when I zoom in



There is a discontinuous derivative near the origin.

It appears that using my k array in the definition of wave = np.exp(- x**2 / (2*σ**2)) * np.exp(1j*k*x) is what is causing the issue. I should not use my k array for this step. I should instead define a specific k for the IC.

If I define (by a brave guess) **k0 = 10 #** oscillations per unit of space, then the error is fixed.



NEXT: The 1D time dependent Schrödinger equation and the RF-CAP (probably I will try V = igx^3 given that this potential is an example in Moiseyev and Bender) I'll consider a simple case with the parameters as follows:

$$\frac{d\Psi(x,t)}{dt} = \left[\frac{i\hbar}{2m}\frac{d^2}{dx^2} + V(x,t)\right]\Psi(x,t) \qquad \text{where } V(x,t) = ix^3 \text{ ie } g=1,\ \varepsilon=1$$

where we can write,

$$\frac{d^2\Psi}{dx} = \mathcal{F}^{-1}\{-k^2\mathcal{F}\{\Psi\}\}$$

from the fourier transform property:

$$\mathcal{F}\{f'(x)\} = ik\int f(x)e^{-ikx}dx = ik\mathcal{F}\{f(x)\}$$

$$\therefore \frac{d\Psi(x,t)}{dt} = \left[\frac{i\hbar}{2m}\mathcal{F}^{-1}\{-k^2\mathcal{F}\{\Psi\}\} + V(x,t)\right]\Psi(x,t)$$

Calculating the FT and inverse FT requires that I import from scipy.fft the functions fft and ifft in my script.
I define my ODE as:

```
# Schrodinger equation (or first order time derivarive)
def Schrodinger_eqn(Ψ, t):
    return 1j * hbar/(2 * m) * ifft((-k**2) * fft(Ψ)) + 1j*x**3 * Ψ
```

Then, I need to define the method of 4th order Runge Kutta for implementing the time evolution of my state.

```
def Runge_Kutta(t, delta_t, Ψ):
    k1 = Schrodinger_eqn(t, Ψ)
    k2 = Schrodinger_eqn(t + delta_t / 2, Ψ + k1 * delta_t / 2)
    k3 = Schrodinger_eqn(t + delta_t / 2, Ψ + k2 * delta_t / 2)
    k3 = Schrodinger_eqn(t + delta_t, Ψ + k3 * delta_t / 2)
    return Ψ + (delta_t / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
```

NEXT: I want to see if it all works!

The steps are:
1. Wrap Runge_kutta in a while loop and feed it my initial gaussian wave packet as Ψ.
2. FIgure out HOW BIG is delta_t? My initial guess is 0.01
3. Define hbar and m in the Schrödinger equation to be equal to 1 to test the numerics.
4. Later start guess work for a smaller system (like an electron wave packet or something)

There is an error in my Schrödinger equation!
Because I didn't expand the factor of (-i/ħ) with the potential term
Therefore this is the corrected Schrödinger equation I will use in my code.

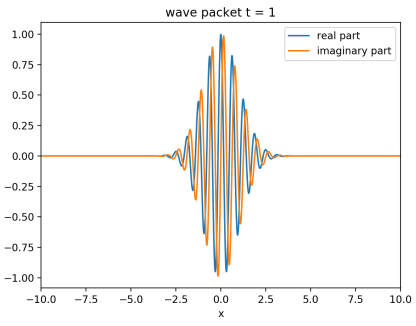
$$\frac{d\Psi(x,t)}{dt} = \frac{-i}{\hbar}\left[\frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + V(x,t)\right]\Psi(x,t)$$

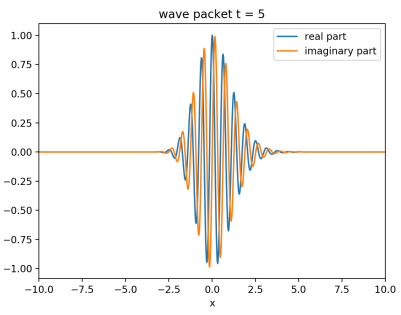
```
# Schrodinger equation (or first order time derivarive)
def Schrodinger_eqn(t, Ψ):
    return (-1j / hbar) * -hbar**2/(2 * m) * ifft((-k**2) * fft(Ψ)) + (-1j / hbar) * 1j*x**3 * Ψ
```

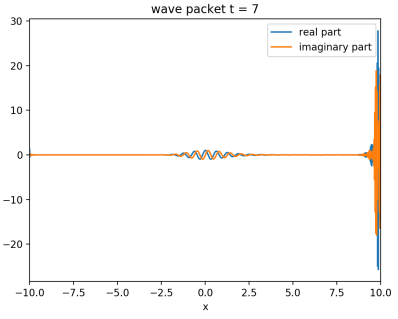
The while loop

I decided to double the x range to make sure that the package had space to evolve and go to zero at the boundary. Therefore x_max = 10

```
i = 0
t = 0
t_final = 1
delta_t = 0.01
m = 1
while t < t_final:

    plt.plot(x, np.real(wave), label="real part")
    plt.plot(x, np.imag(wave), label="imaginary part")
    plt.xlim(-x_max, x_max)
    plt.legend()
    plt.xlabel("x")
    plt.title(f"wave packet t = {i}")

    plt.savefig(folder/f'{i:04d}.png')
    # plt.show()
    plt.clf()

    wave = Runge_Kutta(t, delta_t, wave)
    i += 1
    t += delta_t
```

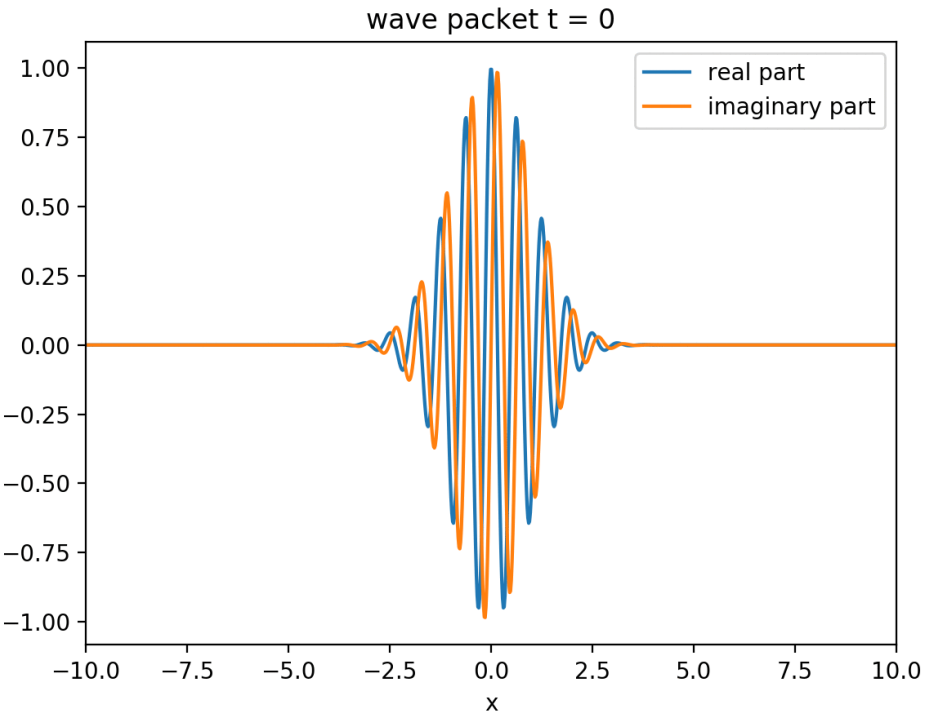






I get some movement of the wave packet to the right but it soon blows up at the boundary! Seems like it still doesn't have enough space to go to zero at the edges and so we get undesirable interference soon after we start evolving in time.

Here is a small gif to demonstrate the blow up…



I use **convert gif_wavepacket/* moving_wave.gif** to make gif on the command line. I only show a few frames here but it can be seen that after frame 8 every other frame is blown up.

I also used **ffmpeg -i "wavepacket_time_evolution/%04d.png" moving_wave.mp4** in the command line to make a little movie with all the frames from the output (careful with those quote marks since we don't want them to be "smart") . Unfortunately I cannot share the movie in this log, since it is an unsupported format.

Debugging the blow up!

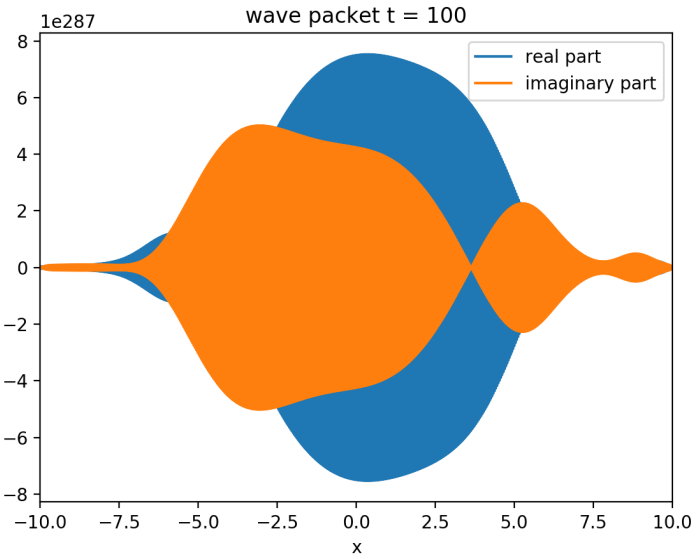
I found an error in the function Runge_Kutta specifically the variable k4. This is now corrected:

```
k4 = Schrodinger_eqn(t + delta_t, Ψ + k3 * delta_t)
```

There also was an error in the definition of my x_step, This is now corrected `x_step = x[1] - x[0]`
After fixing these bugs I still have blow ups on the whole wave packet from frame 5 onwards.

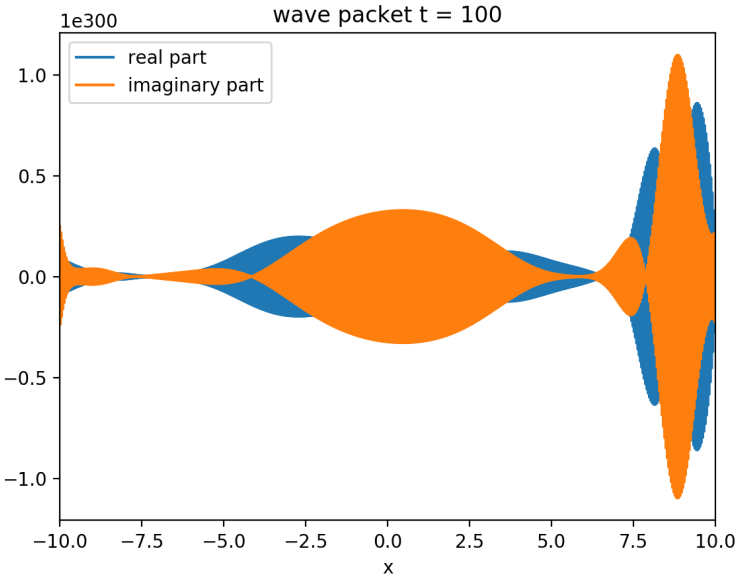
Maybe my delta_t is too large?
While talking to Chris about my problem he brought up the Nyquist frequency. Which is a new term for me.
Delta_t must be small enough to be able to resolve motion of the nyquist mode within the spatial step specified.

After this chat I changed delta_t = 0.001. But A blow up still occurs



What about removing the phase factor from my wave? Ie. only using

```
wave = np.exp(1j*k0*x)
```

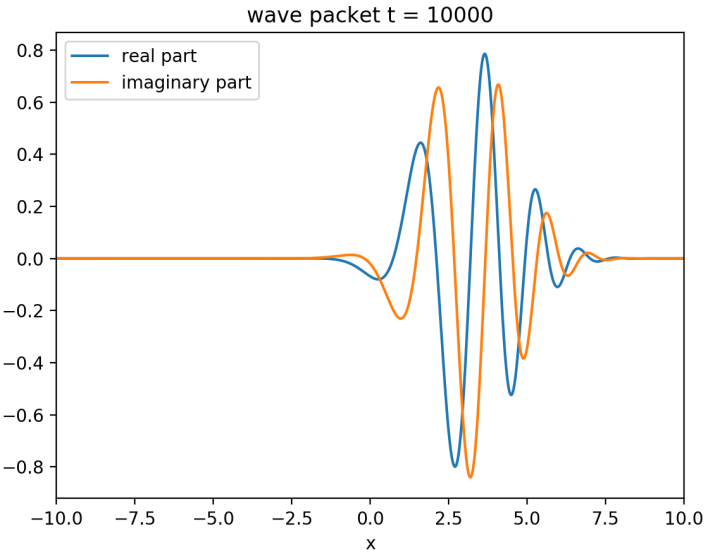


The blow up is different, but still this situation is not ideal.

Based on the possibility of having discontinuous boundaries based on not having an integer number of waves within the region I changed k0 to k0 = 2*pi /x_max * 5

I also decreased the time step, currently delta_t =0.0001

What if my assumption of the potential is the problem?
Obviously this is the main goal of the exercise but I am actually not sure that this is the correct potential to use in my numerics… What happens if I delete the potential ix?



I got a propagating wave packet!
In conclusion: The time step was too large. The **nyquist** mode must not move more than the spatial step otherwise the numerics won't be able to resolve the motion in the simulation.

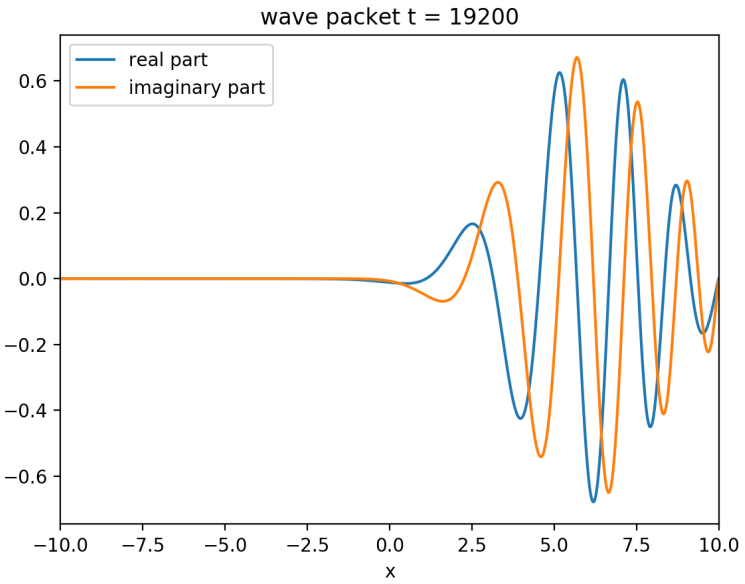
The imaginary potential (RF-CAP) remains to be added to the Schrödinger equation.

14/03/21
Currently, I have continuous BCs, so I don't see non-physical interference effects due to reflections off the edge. To fix this I need to impose a box potential on the edges of my frame.

SQUARE WELL POTENTIAL

```
# Square well potential
sw = np.zeros_like(x)
# depth
sw[0] = sw[-1] = k0**2
```

Is an array that is zero everywhere except at the boundaries. Its height is calculated based on the kinetic energy of the wave packet. It appears that the barrier is not big enough to prevent tunnelling since nothing changed in the simulation so far using this depth! When I increase the barrier depth by 100 I still get tunnelling. Therefore I will increase the barrier depth by 1000. Ie. 1000*k0^2



Yay! I have the desired **undesirable** reflection!

Constructing the RF CAP

I am reading chapter 5 of Moiseyev since apparently this is where the implementation of RF-CAP is discussed. Starting to read this chapter gives me the hibbie-jibbies since the first sentence states "...as discussed in the previous chapter…" and I have not attempted to read chapter 4.
Still, I am going to see if I can get anything out of chapter 5 in order to finish my simulation.
From (Moiseyev 130.131)

5.4 The smooth exterior scaling Transformation

Let us first consider a one-particle three-dimensional Hamiltonian with spherical symmetry. When the total angular momentum is equal to zero this problem reduces to a 1D problem and r can be replaced by x in all equations for the RF-CAP. We begin with the Hamiltonian in the complex variable ρ:

$$\hat{H}(\rho) = -\frac{\hbar^2}{2M}\frac{\partial^2}{\partial\rho^2} + V(\rho)\,. \tag{5.33}$$

Now we define the smooth exterior scaling transformation by choosing a smooth path, $F(r)$, in the complex ρ plane that is as close as possible to the real axis r in the region where the interaction potential $V(r) \neq 0$, i.e., $\rho \cong r$. However,

$$\rho = F(r) \to re^{i\theta} \quad \text{as} \quad r \to \infty\,. \tag{5.34}$$

Next, in order to simplify the expression for the volume element such that it will be equal to $d\rho = dr$ rather than $d\rho = f(r)dr$ we carry out the transformation

$$\hat{H}_f = \hat{H}(r) + \hat{V}_{\text{RF-CAP}}(r)\,, \tag{5.37}$$

where the reflection-free absorbing potential, $\hat{V}_{\text{RF-CAP}}(r)$ contains two contributions,

$$\hat{V}_{\text{RF-CAP}} = \Delta V_f(r) + \hat{V}^{(f)}_{\text{CAP}}\,. \tag{5.38}$$

The first contribution $\Delta V_f(r)$ comes from the difference in the potential $V(\rho)$ along $F(r)$ and along the real axis,

$$\Delta V_f(r) = V(F(r)) - V(r)\,. \tag{5.39}$$

This term vanishes when we are dealing with a short-range potential and the scaling is carried far enough from the relevant region of interaction of the potential. The second contribution to $\hat{V}^{(f)}_{\text{CAP}}$ comes from the scaling of the kinetic energy operator

and is independent of the analyzed potential, thus producing an effective absorbing potential given by

$$\hat{V}^{(f)}_{\text{CAP}} = V_0(r) + V_1(r)\frac{\partial}{\partial r} + V_2(r)\frac{\partial^2}{\partial r^2}\,, \tag{5.40}$$

where

$$V_0(r) = \frac{\hbar^2}{4M}f^{-3}(r)\frac{\partial^2 f}{\partial r^2} - \frac{5\hbar^2}{8M}f^{-4}(r)\left(\frac{\partial^2 f}{\partial r^2}\right)^2\,, \tag{5.41}$$

$$V_1(r) = \frac{\hbar^2}{M}f^{-3}(r)\frac{\partial f}{\partial r}\,, \tag{5.42}$$

$$V_2(r) = \frac{\hbar^2}{2M}(1 - f^{-2}(r))\,. \tag{5.43}$$

Now we can introduce the smooth-exterior-scaling (SES) Hamiltonian as

$$\hat{H}_{\text{SES}} \equiv \hat{H}_f = \hat{T}_r + \hat{V}^{(f)}_{\text{CAP}}\,, \tag{5.44}$$

where $\hat{T}_r$ is the unscaled kinetic energy operator as defined in the standard (Hermitian) formalism of quantum mechanics.

I do not understand many terms and concepts used in this method. I probably can't finish this task without having a look at previous chapters.

Important points

- IF a Hamiltonian has unbroken $\mathcal{PT}$- symmetry then it has real eigenvalues provided the parameter g has values smaller than a critical value. Above this value the spectrum of these Hamiltonians is comprised of real and complex conjugate pairs.(Moiseyev 11)
- Bender mentions the ε parameter as a key to the number of real eigenvalues in the real part of the energy spectrum of a Hamiltonian.
- There is no experimental evidence that proves definitely that quantum systems defined by $\mathcal{PT}$- symmetric NHH do exist in nature (Bender 953)
- I don't have to read and answer every single question… progress is not linear (chill)

Important questions

- Do eigenfunctions of NH operators for a complete set in the series expansion of any wave packet?
  Answer lies in CHAPTER 5 of Moiseyev
- Are different poles of the S-matrix orthogonal to each other?
- Are the expectation values of the physical quantities within the NHQM formalism real or complex?

Week 3 (15/03/21 – 21/03/21)

Aims:

1. Try to understand Gao et al.'s (2x2 Hamiltonian)
   (Compare Gao et al. to Bender page 986 to see if comparison is useful)
2. Reproduce Fig 1 in Bender
3. **Simulate a RF-CAP**

Tasks:

1. ~~Meeting with J&M (Wednesday 14:30)~~
   ~~inform J&M about my **RF-CAP** situation.~~
2. ~~Attend group meeting~~

16/03/21

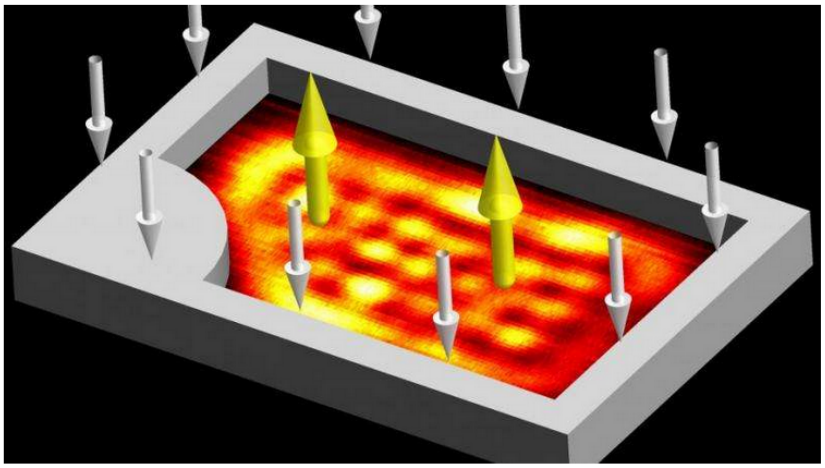
Observation of non Hermitian degeneracies in a chaotic exciton polariton billiard

Notes, direct quotes and screenshots from Gao et al. :

- Exciton-polaritons (E-P) are a non Hermitian system
  (requires constant energy pumping and continuously decays releasing coherent radiation)
- non-Hermicity modifies the structure of **modes** and **spectral degeneracies of E-P**
  (affecting quantum transport, localisation and dynamics)
- **Exceptional points (EP):** non hermitian spectral degeneracies of the modes
  Consider reading : doi:10.1088/1751-8113/45/44/444016
- Topological berry phase
  (By going in a loop around the exceptional point in the parameter plane, you can never return to the same quantum state you started from. A second loop will then flip the state upside down. (ANU))

Metastable states (decay in time) ⇒ complex eigenenergies

2D Quantum billiard (modified Sinai billiard) results in strongly correlated energy levels and transition to quantum chaos. The billiard used has "soft" walls (this is generically adequate for observing mixed regular-chaotic behaviour) The observation of multiple degeneracies is expected in the "hard" wall setup.
The increase in the observed number of degeneracies implies the transition from regular to chaotic dynamics



Credit: ANU Polariton BEC laboratory.

The observed energy levels crossing and anti-crossing in the vicinity of degeneracies (as expected from the non- Hermitian system) is because the energy eigenvalues are complex.

The REAL and IMAGINARY PARTS of the eigenvalues correspond to the real energies and **bandwidth of the modes** respectively

Important note: provided that the internal area remains unchanged the thickness of the walls (*d)* does not affect the geometry of the billiard and primarily controls the imaginary part of the non-Hermitian potential barrier (V'').
Given the prior explanation I am unsure about FIg 2 showing one pair of modes near a degeneracy and why they can cross and anticross depending on the value of the parameter *d.* From (Gao et al. 556):

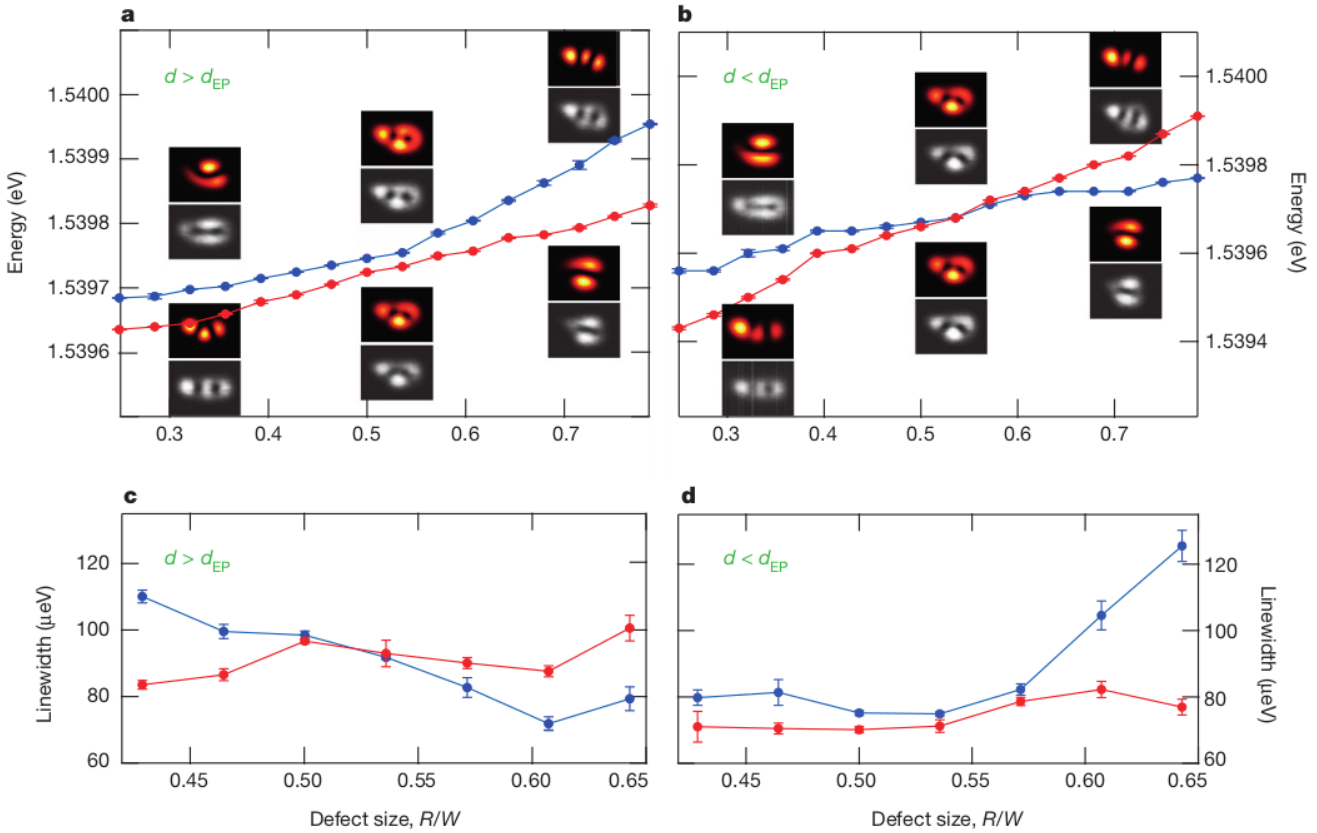


Figure 2 | Crossing and anti-crossing for two near-degenerate modes. These modes are boxed in Fig. 1c, d. **a**–**d**, Experimentally observed anti-crossing (**a**) and crossing (**b**) of eigenenergies of two modes in the spectrum of the exciton-polariton Sinai billiard with varying parameter R (see Fig. 1) for thick, $d \approx 6\,\mu m$ (**a**, **c**), and thin, $d \approx 4\,\mu m$ (**b**, **d**), billiard walls; d_{EP} is the value corresponding to the exceptional point. Panels **c** and **d** show the corresponding crossing and anti-crossing of the linewidths (that is, imaginary parts of the complex eigenvalues). The error bars in **a**–**d** originate from numerical fitting of the spectroscopic data (see Methods). The upper (lower) inset panels in **a** and **b** illustrate the numerically calculated (experimentally imaged) spatial structure of the eigenmodes at different values of the parameter R. Details of the hybridization region are given in Methods.

For two near degenerate modes:

if $d < d_{EP}$:

The Energy (real part) and Linewidth (imaginary part) anti-cross and cross respectively

elif $d > d_{EP}$:

The Energy (real part) and Linewidth (imaginary part) cross and anti-cross respectively

Does this mean that the exceptional point forms at d_{EP}? Yes.

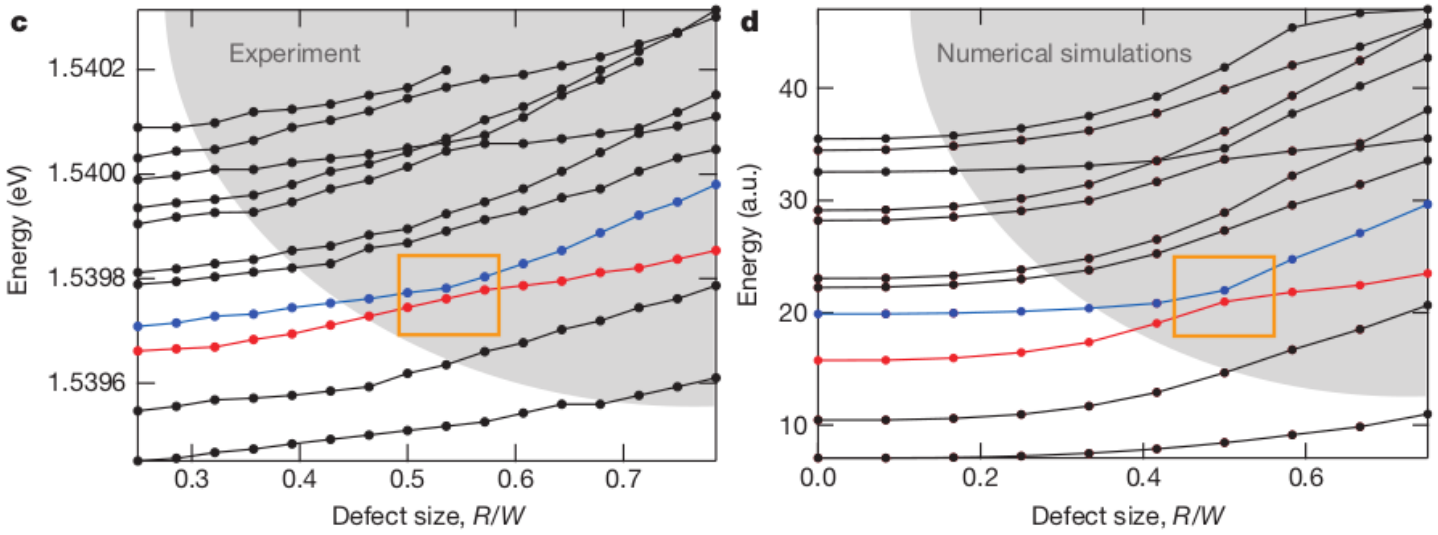
17/03/21
MEETING Day!
The following text includes some direct quotes from (Gao et al. 554):

The main properties of the eigenstate of the E-P condensate can be described by a linear Schrödinger equation with a complex 2D potential $V(r) = V'(r) + iV''(r)$

Where the real part of the potential $V'(r) \propto P(r)$ is the potential barrier (Sinai billiard boundary with a gaussian envelope).The parameter R mostly affects the real part of the potential and hence the energy difference between the modes. The optical pump rate $P(r)$is induced by the strong repulsive interaction between the excitonic reservoir populated by the pump and the polariton BEC.

The imaginary part of the potential $V''(r) \propto P(r) - \gamma$combines the gain profile introduced by the optical pump $P(r)$with the spatially uniform loss γ due to polariton decay.

Direct quotes and **screenshots** from (Gao et al. 555):
Modifying the Radius defect of the potential R varies the geometry of the billiard tuning the eigenmodes at different rates, Ie. some energy levels approach each other with varying values of R (hence they observe regular to chaos dynamics transitions)



(see Methods). **c**, **d**, Experimentally measured (**c**) and numerically simulated (**d**) spectra $E(R/W)$ for the first 11 modes of the billiard in arbitrary units (a.u.). With growing R, numerous degeneracies and quasi-degeneracies proliferate in the grey area, which is a signature of the transition to quantum chaos in the Hermitian Sinai billiard[7]. Topological properties of two near-degenerate modes (red and blue in the orange rectangles) are analysed in detail in Figs 2–4.

The grey area shows both behaviours crossing and anti-crossing of the energy levels near degeneracies

So I think that I figured out the function of parameter *d:*
The real and imaginary parts of the complex eigenenergies of the NHH may cross or anti-cross depending on control parameter *d* .
Thicker walls imply less decay. Which implies more energy in the system.

DEGENERACY REGION

The behaviour of two billiard modes in the vicinity of a degeneracy can be described by a two-level system with an effective coupling .The corresponding NHH is:

$$\hat{H} = \begin{pmatrix} \tilde{E}_1 & q \\ q^* & \tilde{E}_2 \end{pmatrix}, \quad \tilde{E}_{1,2} = E_{1,2} - i\Gamma_{1,2}$$

Where $\tilde{E}_{1,2}$ are the complex energies of two uncoupled modes. $\Gamma_{1,2}$are decay/gain rates. Coupling between the modes is$q$ with q^* as the complex conjugate.

Other important expressions

1. The mean complex energy: $\tilde{E} = (\tilde{E}_1 + \tilde{E}_2)/2 \equiv E - i\Gamma$
2. The complex energy difference: $\delta\tilde{E} = (\tilde{E}_1 - \tilde{E}_2)/2 \equiv \delta E - i\,\delta\Gamma$
3. Eigenvalues:$\lambda_{1,2} = \tilde{E} \pm \sqrt{\delta\tilde{E}^2 + |q|^2}$

Exceptional points

EP are non- Hermitian spectral degeneracies (eigenvalues coalesce) , that can cause remarkable wave phenomena:

- unidirectional transport
- anomalous lasing /absorption
- chiral modes

Eigenstates also coalesce at the EPs (where $i\delta\tilde{E} = \pm |q|$) and form a single **chiral mode**.

Gao et al. also assume$q$ is fixed EPs appear in the parameter plane as $(\delta\tilde{E}_{EP}, \delta\Gamma_{EP}) = (0, \pm |q|)$ **and** $\delta\Gamma > 0$**so that there is only one EP in the domain of interest.**

From (Gao et al. 556):

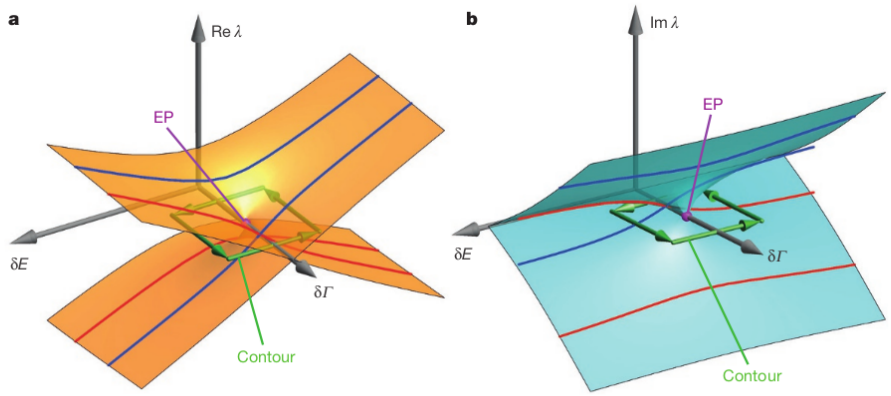


Figure 3 | Eigenvalues of a two-level non-Hermitian model in the vicinity of the exceptional point. **a**, **b**, Real (**a**) and imaginary (**b**) parts of the eigenvalues $\lambda_{1,2}$ of the model (equation (1)) as functions of two parameters, δE and $\delta\Gamma$. The exceptional point (EP) is shown in magenta. The crossing and anti-crossing of the real and imaginary parts of the eigenvalues as functions of δE, for $\delta\Gamma < \delta\Gamma_{EP}$ and $\delta\Gamma > \delta\Gamma_{EP}$, are shown in red and blue. This is in correspondence with the experimentally observed behaviour in Fig. 2. Traversing along the green contour encircling the exceptional point in the $(\delta E, \delta\Gamma)$ plane reveals the non-trivial topology of eigenmodes, as shown in Fig. 4.

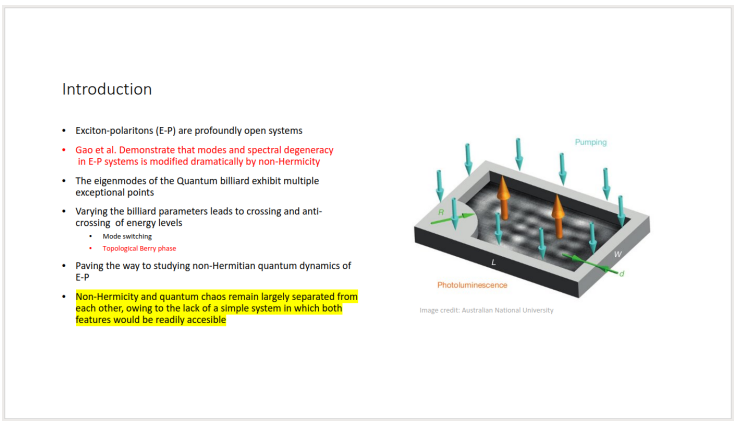
Correspondence of the parameters in the model $(\delta E, \delta\Gamma\ )$ to the parameters in the E-P billiard $(R,\ d)$

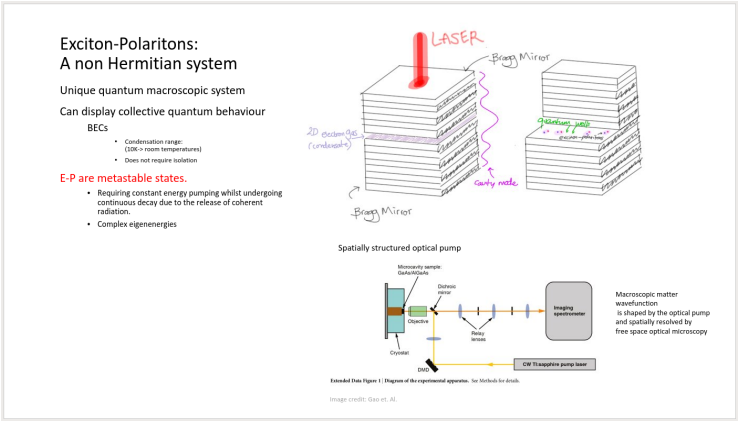
- Increasing R (corresponds to a greater confinement)⇒increasing δE
- Thickness of billiard walls controls gain and loss profile $V''(r)$

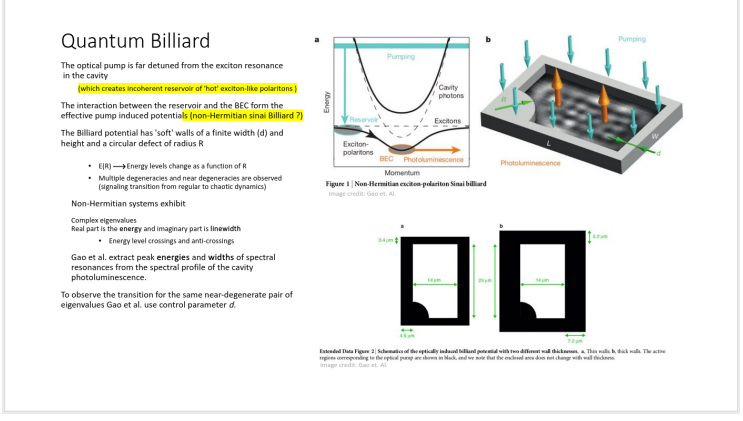
MEETING notes:
J&M asked me to **present this paper** to them in our next meeting.
They also suggested that I slow down on my RF-CAP simulation and read as much as possible in order to figure out how to implement the absorbing potential in my equation. Let's see how far I get!

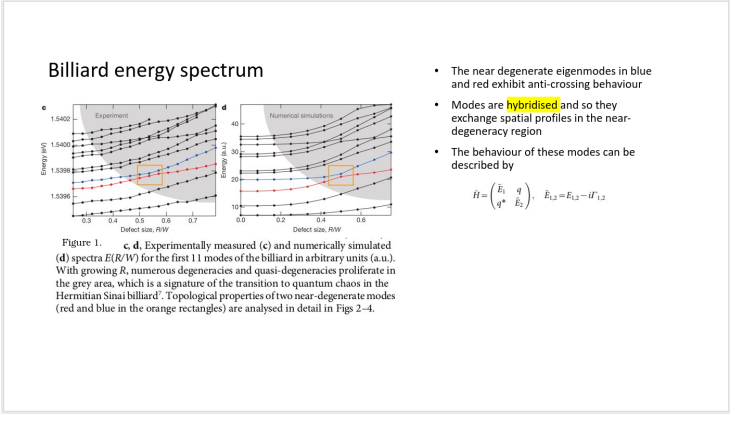
J&M found this paper https://arxiv.org/abs/2006.01837 which might be of use in my understanding of NHQM.

18/03/21
Today I re-read the first couple of pages of Gao et al.'s paper and started to work on some slides for my presentation on the next meeting. This is not the final form of the slides but I guess this process has helped me to try to map my ideas a little more concisely.









I currently don't understand the following terms:

1. **Hybridization**
   Applied to the cavity and the modes
2. **Lobes**
   The modes can have lobes (horizontal and vertical)
3. $i\delta\tilde{E}_{EP} = \pm\ |q|$(coalesced eigenstates condition)

Important points:

- I think that the paper aims to provide a link between a non-Hermitian system and quantum chaos
- An exceptional point happens when two different modes of a non hermitian system coalesce into one. (ANU)
- Topological Berry phase: By going in a loop around the exceptional point in the parameter plane, you can never return to the same quantum state you started from. A second loop will then flip the state upside down. (ANU)

Important questions:

- What are the **modes** of the exciton-polariton systems?
  The eigenmodes are the **wavefunctions** (Gao et al. 555)
  So, the eigenmodes = eigenstates ie. mode = state
- Why are the energy levels in a quantum billiard strongly correlated?
- Is the linewidth (imaginary part of eigenvalues) the same as the spatial structure of the eigenmodes?

Week 4 (22/03/21 – 28/03/21)

Aims:

1. **Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)**
2. Reproduce Fig 1 in Bender
3. Simulate a RF-CAP

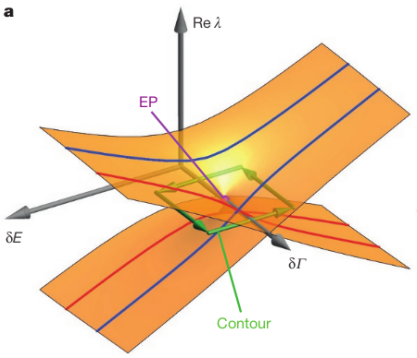
Tasks:

1. ~~Finish up slides for billiard talk~~
2. ~~Meeting with J&M (Wednesday 14:15)~~
   ~~give the billiard talk!~~
3. ~~START PROGRESS REPORT~~
   ~~What to write about?~~

22/03/21

Topological Berry phase

Gao et al. observe a non- trivial topological defect known as an exceptional point (EP).They trace the topological structure of two modes in the vicinity of the EP and compare the eigenmodes along a contour. They loop around the parameter space (by modifying the biliard parameters (*R, d*)). In the following image (from Gao et al.) I can see that the eigenstates change with the parameters. The paper explains that varying *R* mostly affects the real part of the potential which corresponds to an increase in the energy (δE) when they increase *R*.



Modifying the wall thickness *d* controls the gain/loss (dissipation parameter) profile of the potential $\delta\Gamma$.

Increasing *d* corresponds to decreasing $\delta\Gamma$

Different modes have different spatial overlaps characterised by different integral (spatially averaged) dissipation parameters $\Gamma_{1,2}$. The effective coupling *q* is determined by the spatial overlap between two modes away from the hybridization region

In the figure above (part of fig 3. Gao et al) it can be seen that the structure of the complex eigenvalues reveals a branching point in the vicinity of the EP. Following the green contour around the EP I can see that it is necessary to do the loop twice in order to return to the starting point in the space since a single loop takes us through the branch. Most significantly once we return to the starting point we have a topological phase shift of π.

The topological Berry phase is this phase shift. It occurs because of the encircling of a non-Hermitian degeneracy in a 2D parameter space

Gao et al. do not consider adiabatic evolution of modes due to the variation of parameters (R, d) in time. They explain that such evolution would be accompanied by non-avoidable adiabatic transitions in the non-Hermitian case. They instead examine the topological structure and geometrical connection of stationary modes depending on the parameter values. (?)

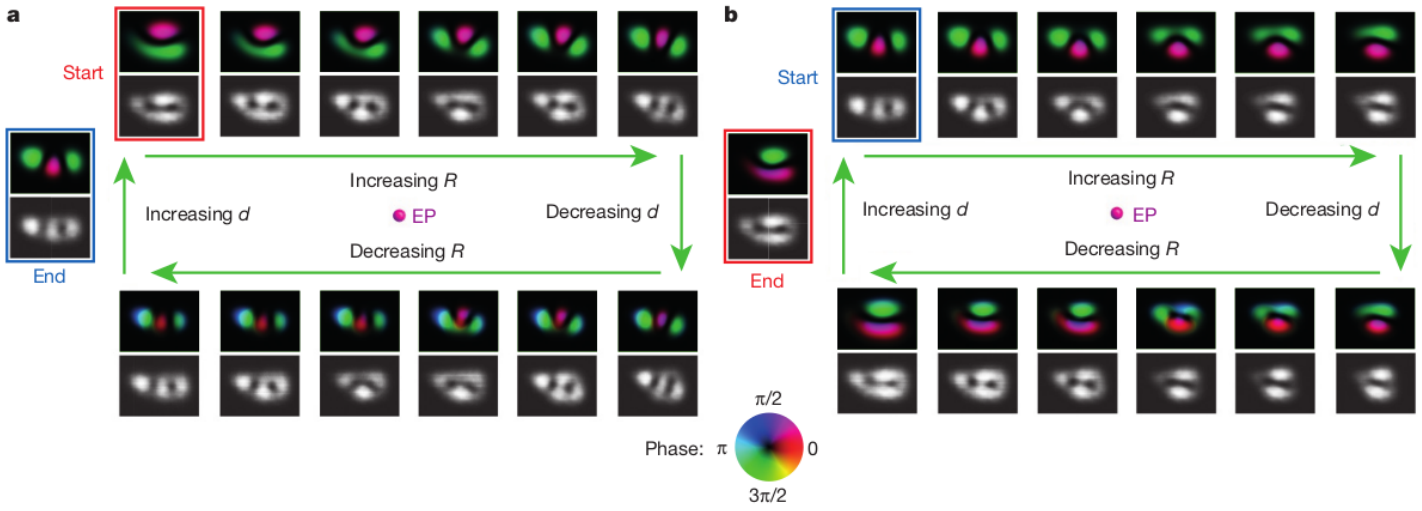
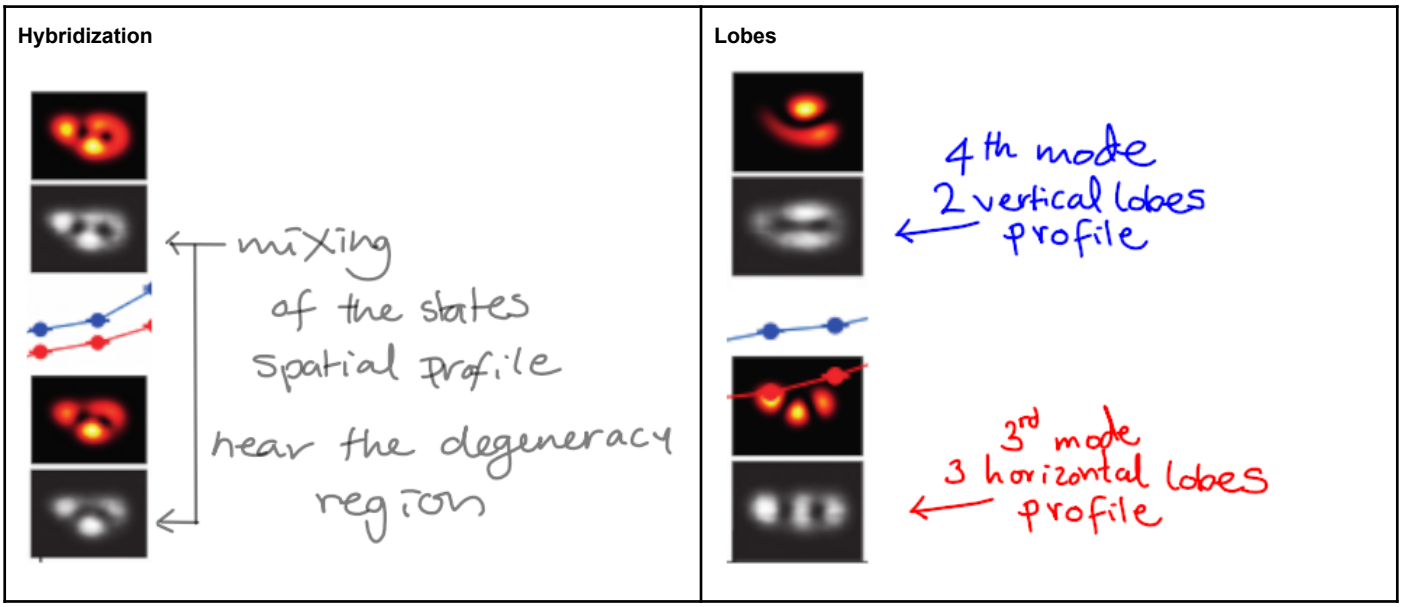


Figure 4 | Observation of the topological Berry phase acquired after circling around the exceptional point in the parameter plane. Transmutations of spatial distributions (black-and-white panels) of the selected eigenmode (from the pair shown in Fig. 2) along the closed contour in the parameter space $(R, d) \sim (\delta E, \delta\Gamma)$ encircling the exceptional point (see Fig. 3). Parameters are not varied in time during the measurements, and each distribution corresponds to the stationary mode at the corresponding parameter values. **a**, **b**, The first loop (**a**) shows the transition to a different branch (mode) through the hybridization region (see explanations in text); the second loop (**b**) returns the mode to the original one with a π topological phase shift[23,24]. The phases (colour panels) are inferred from comparison with the numerically calculated modes. The modes corresponding to the 'start' and 'end' points of the loop on the red (blue) branch in Figs 2a, b and 3a are boxed in red (blue).

23/03/21

I have been correcting my slides and reading the methods section of the paper. I am still not done with the work but I think I understand a lot more than a week ago.



But I am still confused about the meaning of this expression

$i\delta\hat{E}_{EP} = \pm\ |q|$(coalesced eigenstates condition)

24/03/21
MEETING Day! This is the final form of the slides:

Introduction

- Gao et al. Demonstrate that modes and spectral degeneracy in E-P systems is modified dramatically by non-Hermicity
- The eigenmodes of the **Quantum billiard** exhibit multiple exceptional points
- Varying the billiard parameters leads to
  - o Crossing and anti-crossing of energy levels
  - o Mode switching
  - o Topological Berry phase

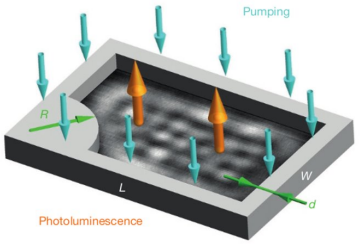


Image credit: Gao et. Al.

Exciton-Polaritons

Unique quantum macroscopic system

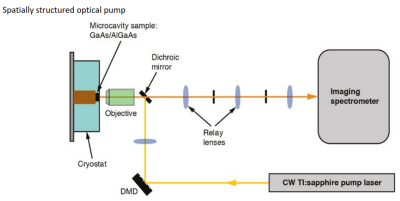
E-P and metastable states.

- Requiring constant energy pumping whilst undergoing continuous decay due to the release of coherent radiation.
- Complex eigen energies

Can display collective quantum behaviour

BECs

- Condensation range: (10K-> room temperatures)
- Condensate does not require isolation



Extended Data Figure 1 | Diagram of the experimental apparatus. See Methods for details.

Image credit: Gao et. Al.

Macroscopic matter wavefunction is shaped by the optical pump and spatially resolved by free space optical microscopy

Quantum Billiard

Microcavity specs

i. Semiconductor sample: GaAs, AlGasAs
ii. 12 Quantum wells (~13nm wide each)
iii. QW distributed in 3 sets of 4 at antinodes
iv. Sandwiched between 32/36 Bragg mirror pairs
v. The optical cavity pump is far detuned from the exciton resonance
vi. Sample is maintained at 5.6 K

BEC is formed by a quasi-continuous off resonant linearly polarised pump beam

DMD shapes and images the billiard as a spatial pump profile

The billiard has 'Soft' walls of a finite width *d* and a circular defect of radius *R*

The pump creates an incoherent reservoir of 'hot' exciton-like polaritons

Peak **energies** and **widths** are extracted from the spectral profile of the cavity photoluminescence (CCD camera and spectrometer)

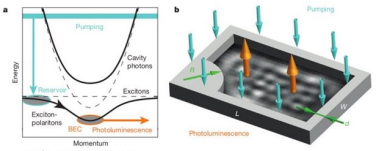
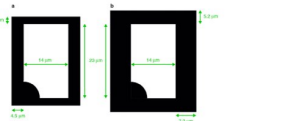


Figure 1 | Non-Hermitian exciton-polariton Sinai billiard

Image credit: Gao et. Al.



Extended Data Figure 2 | Schematics of the optically induced billiard potential with two different wall thicknesses. **a**, Thin walls; **b**, thick walls. The active regions corresponding to the optical pump are shown in black, and we note that the enclosed area does not change with wall thickness.

Image credit: Gao et. Al.

Billiard energy spectrum

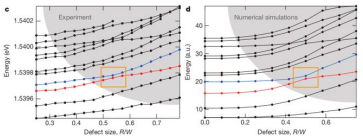


Figure 1. **c, d**, Experimentally measured (**c**) and numerically simulated (**d**) spectra $E(R/W)$ for the first 11 modes of the billiard in arbitrary units (a.u.). With growing R, numerous degeneracies and quasi-degeneracies proliferate in the grey area, which is a signature of the transition to quantum chaos in the Hermitian Sinai billiard[7]. Topological properties of two near-degenerate modes (red and blue in the orange rectangles) are analysed in detail in Figs 2–4.

Image credit: Gao et. Al.

- The near degenerate eigenmodes in blue and red exhibit anti-crossing behaviour
- Modes are hybridised
- The behaviour of these modes can be described by the non hermitian Hamiltonian

$$\hat{H} = \begin{pmatrix} \hat{E}_1 & q \\ q^* & \hat{E}_2 \end{pmatrix}, \quad \hat{E}_{1,2} = E_{1,2} - i\Gamma_{1,2}$$

The mean complex energy: $\hat{E} = (\hat{E}_1 + \hat{E}_2)/2 \equiv E - i\,\Gamma$

The complex energy difference: $\delta\hat{E} = (\hat{E}_1 - \hat{E}_2)/2 \equiv \delta E - i\,\delta\Gamma$

Eigenvalues: $\lambda_{1,2} = \hat{E} \pm \sqrt{\delta\hat{E}^2 + |q|^2}$

The complex eigenvalues and eigenstates coalesce at the **exceptional points.**

Exceptional points (EPs)

Non- Hermitian spectral degeneracies

Assuming q is fixed, The eigenstates coalesce at the EPs ($i\delta\hat{E} = \pm|q|$) and form a single **Chiral mode.**

EPs appear in the **parameter plane as** ($\delta\hat{E}_{EP}, \delta\Gamma_{EP}$) = $(0, \pm|q|)$ **so that there is only one EP in the domain of interest.**

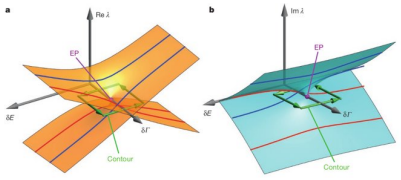


Figure 3 | Eigenvalues of a two-level non-Hermitian model in the vicinity of the exceptional point. **a**, **b**, Real (**a**) and imaginary (**b**) parts of the eigenvalues $\lambda_{1,2}$ of the model (equation (1)) as functions of two parameters, $\delta\hat{E}$ and $\delta\Gamma$. The exceptional point (EP) is shown in magenta. The crossing and anti-crossing of the real and imaginary parts of the eigenvalues as functions of δE, for $\delta\Gamma < \delta\Gamma_{EP}$ and $\delta\Gamma > \delta\Gamma_{EP}$, are shown in red and blue. This is in correspondence with the experimentally observed behaviour in Fig. 2. Traversing along the green contour encircling the exceptional point in the $(\delta E, \delta\Gamma)$ plane reveals the non-trivial topology of eigenmodes, as shown in Fig. 4.

Image credit: Gao et. Al.

Experimental data: Anti-crossing (a) and crossing (b) of two near-degenerate modes

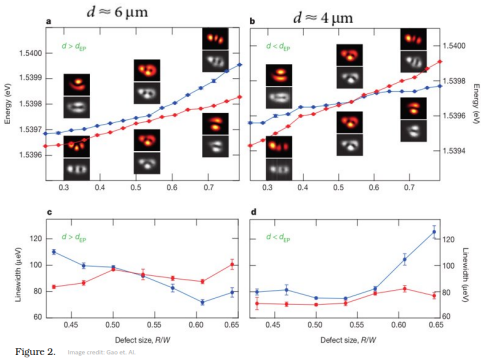


Figure 2. Image credit: Gao et. Al.

- Eigen energies of two modes as functions of R
- Wall thickness ***d*** **varies**
- Where d_{EP} is the value corresponding to the exceptional point
- Inset panels correspond to spatial structure of the eigenmodes varying with R

Topological Berry phase

The complex eigenvalues structures reveal a branching point in the vicinity of the EP.

It is necessary to loop around the EP twice in order to return to the starting point in the space since a single loop takes us through the branch.

Once we return to the starting point we have a topological phase shift of π.

Figure 4 depicts the Berry phase in the simulated phase profiles

The work of Gao et al demonstrates the creation of highly controllable non-Hermitian exciton polariton potentials.

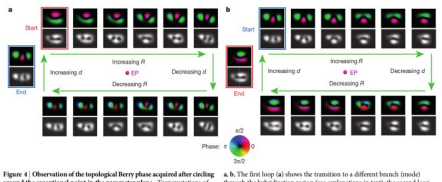


Figure 4 | Observation of the topological Berry phase acquired after circling around the exceptional point in the parameter plane. Transmutations of spatial distributions (black-and-white panels) of the selected eigenmode (from the pair shown in Fig. 2) along the closed contour in the parameter space $(R, d) - (\delta E, \delta\Gamma)$ encircling the exceptional point (see Fig. 3). Parameters are not varied in time during the measurements, and each distribution corresponds to the stationary mode at the corresponding parameter values.

a, **b**, The first loop (**a**) shows the transition to a different branch (mode) through the hybridization region (see explanations in text); the second loop (**b**) returns the mode to the original one with a π topological phase shift[14,15]. The phases (colour panels) are inferred from comparison with the numerically calculated modes. The modes corresponding to the 'start' and 'end' points of the loop on the red (blue) branch in Figs 2a, b and 3a are boxed in red (blue).

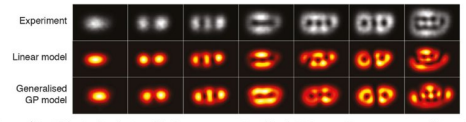
Image credit: Gao et. Al.

Billiard Modelling

The full dynamics of the E-P condensate in Billiard potential can be described by the generalised complex Gross-Pitaevskii equation for the condensate wave function

$$i\hbar\frac{\partial\psi(\mathbf{r},t)}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + (g - i\gamma_{nl})|\psi|^2 + (g_R + i\hbar R)n_R(\mathbf{r}) - i\hbar\gamma\right]\psi + i\hbar\Re[\psi(\mathbf{r},t)] \quad (2)$$

Strength Polariton–Polariton interaction; Strength Reservoir–BEC interaction; reservoir density; Polariton decay rate; effective mass of E-P; gain saturation; Stimulated scattering rate (into BEC); phenomenological energy relaxation



Extended Data Figure 4 | Spatial density distribution of the first seven simultaneously populated lowest-energy modes of the Sinai billiard. Spatial density distributions were obtained from the thick-wall setup (Extended Data Fig. 2b) with $R/W = 0.35$. Top row, experimentally imaged; middle row, calculated using the effective linear potential model; bottom row, calculated using the full dynamical model given by equation (2).

Image credit: Gao et. Al.

Meeting notes:

I think that the main aim now is to understand equation (1).

This is the 2 level linear model used in the paper. It looks considerably easier than the G-P model and somehow it appears very useful.

Aim 1 in my weekly list seems more important than ever!

Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)

What do I write in my progress report?

1. What is my topic?
2. What have I done so far?
   -READING
   -What do I currently understand?
3. What Ideas do I have?
4. What is the goal of my project?

   Why are NHHs useful and for what kind of systems they can be used

27/03/21

Investigation: Linear coupled-mode model in Gao et al.

Gao et al. explain in the **methods** section of their paper that in the degeneracy region, the behaviour of any two modes (*n, n'*) of the billiard potential can be described by a standard (dimensionless) **coupled-mode model**:

$$i\frac{\partial \psi_{n,n'}(\boldsymbol{r},t)}{\partial t} = \left[-\nabla^2 + V'(\boldsymbol{r}) + iV''(\boldsymbol{r})\right]\psi_{n,n'} + \Omega\psi_{n',n}$$

Where Ω characterises the coupling strength between the modes.

Gao et al. define an arbitrary state as separable ie. $\psi_n = a_n(t)\varphi_n(\boldsymbol{r})$. Where φ_n are the basis vectors of the real part of the Hamiltonian H_0 (with real eigenvalues away from the degeneracy)

The imaginary parts of the eigenenergies are given by the overlap between the billiard modes and the exciton reservoir.

The decay and gain rates from the reservoir BEC interaction are $\Gamma_n \propto \int V''(\boldsymbol{r})|\varphi_n(\boldsymbol{r})|^2 \mathrm{d}^2\boldsymbol{r}$

I write the vector equation as:

$$i\frac{\partial}{\partial t}\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix} = \underbrace{\left[-\nabla^2 + V'(r)\right]}_{H_0} + iV''(r)\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix} + \begin{bmatrix}0 & \Omega\\ \Omega & 0\end{bmatrix}\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix}$$

let $H_0 = \left[-\nabla^2 + V'(r)\right]$

$$\therefore\ i\frac{\partial}{\partial t}\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix} = \tilde{E}_{n,n'}\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix} + iV''(r)\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix} + \begin{bmatrix}0 & \Omega\\ \Omega & 0\end{bmatrix}\begin{bmatrix}\Psi_n\\ \Psi_{n'}\end{bmatrix}$$

Real energies (away from degeneracy)

overlap between modes & reservoir

modes coupling

My derivation of equation 1 in the main text:

Obtaining matrix elements :

(1,1) Projecting $|\Psi_{n,n'}\rangle$ onto $|\Psi_{n,n'}\rangle$:

$$\langle\Psi_{n,n'}|i\frac{\partial}{\partial t}|\Psi_{n,n'}\rangle = \langle\Psi_{n,n'}|E_{n,n'}|\Psi_{n,n'}\rangle + \langle\Psi_{n,n'}|iV''(r)|\Psi_{n,n'}\rangle + \langle\Psi_{n,n'}|\Omega|\Psi_{n',n}\rangle$$

$$= E_{n,n'} + i\Gamma_{n,n'} \quad \leftarrow \quad i\propto\Gamma_{n,n'} \text{ with } \propto\in\mathbb{R}\ ?$$

(1,2) Projecting $|\Psi_{n',n}\rangle$ onto $|\Psi_{n,n'}\rangle$:

$$\langle\Psi_{n,n'}|i\frac{\partial}{\partial t}|\Psi_{n',n}\rangle = \langle\Psi_{n,n'}|E_{n',n}|\Psi_{n',n}\rangle + \langle\Psi_{n,n'}|iV''(r)|\Psi_{n',n}\rangle + \langle\Psi_{n,n'}|\Omega|\Psi_{n,n'}\rangle$$

$$= \Omega$$

(2,1) Projecting $|\Psi_{n,n'}\rangle$ onto $|\Psi_{n',n}\rangle$:

$$\langle\Psi_{n',n}|i\frac{\partial}{\partial t}|\Psi_{n,n'}\rangle = \langle\Psi_{n',n}|E_{n,n'}|\Psi_{n,n'}\rangle + \langle\Psi_{n',n}|iV''(r)|\Psi_{n,n'}\rangle + \langle\Psi_{n',n}|\Omega|\Psi_{n',n}\rangle$$

$$= \Omega$$

(2,2) Projecting $|\Psi_{n',n}\rangle$ onto $|\Psi_{n',n}\rangle$:

$$\langle\Psi_{n',n}|i\frac{\partial}{\partial t}|\Psi_{n',n}\rangle = \langle\Psi_{n',n}|E_{n,n'}|\Psi_{n',n}\rangle + \langle\Psi_{n',n}|iV''(r)|\Psi_{n',n}\rangle + \langle\Psi_{n',n}|\Omega|\Psi_{n,n'}\rangle$$

$$= E_{n,n'} + i\Gamma_{n,n'}$$

Hence I derived equation (1),

$$H = \begin{bmatrix} E_{n,n'} + i\Gamma_{n,n'} & \Omega \\ \Omega & E_{n,n'} + i\Gamma_{n,n'} \end{bmatrix} = \begin{bmatrix} \tilde{E}_{n,n'} & \Omega \\ \Omega & \tilde{E}_{n',n} \end{bmatrix}$$

$$\hat{H} = \begin{pmatrix} \tilde{E}_1 & q \\ q^* & \tilde{E}_2 \end{pmatrix}, \quad \tilde{E}_{1,2} = E_{1,2} - i\Gamma_{1,2}$$

Where $\tilde{E}_{n,n'}$ are the complex energies of two uncoupled modes. $\Gamma_{n,n'}$ are decay/gain rates.

Nevertheless, coupling between the modes should be q with q^* as the complex conjugate. And I still have coupling strength Ω

Notes on equation (1) derivation:

- What is the proportionality constant of the imaginary terms in the main diagonal?
- I am unsure about the sign of the gain/decay rates in the main diagonal, since the value reported in the paper is:

$$\tilde{E}_{n,n'} = E_{n,n'} - i\Gamma_{n,n'}$$

- I also don't understand the relation between Ω and q
  - q characterizes the coupling between the states n and n'.
  - Ω characterizes the coupling strength between these two modes.

- The methods section explains that the off-diagonal matrix elements in equation (1) are determined by the degree of spatial overlap between the two modes,

$$q \propto \int \varphi_n^*(\mathbf{r})\varphi_{n'}(\mathbf{r})\,\mathrm{d}^2\mathbf{r}$$

I don't understand this proportionality relation.
Is this a typo? I suspect it because n and n' are supposed to be orthogonal

$$\Leftrightarrow \langle\varphi_n|\varphi_{n'}\rangle = 0$$

28/03/21

Gao et al.'s (2x2 Hamiltonian) vs Bender (page 986)
"Illustrative 2 × 2 matrix example of a PT-symmetric Hamiltonian"

We consider the 2x2 Hamiltonian matrix H in Gao et al.
where the three parameters $E_{1,2}, \Gamma_{1,2}, q$ are real

STEPS:

1. FIND if H is PT symmetric

$H = \begin{bmatrix} \tilde{E}_{1,2} & q \\ q^* & \tilde{E}_{2,1} \end{bmatrix}$ is H PT-symmetric?

PT-symmetry

$H^{PT} = (PT)H(PT) \Leftrightarrow [PT, H] = 0$

where: $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $T \rightarrow$ complex inversion

$\therefore (PT)H - H(PT) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} T \begin{bmatrix} \tilde{E}_{1,2} & q \\ q^* & \tilde{E}_{2,1} \end{bmatrix} - \begin{bmatrix} \tilde{E}_{1,2} & q \\ q^* & \tilde{E}_{2,1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} T$

$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} T \begin{bmatrix} E_{1,2} - i\Gamma_{1,2} & q \\ q^* & E_{2,1} - i\Gamma_{2,1} \end{bmatrix} - \begin{bmatrix} E_{1,2} - i\Gamma_{1,2} & q \\ q^* & E_{2,1} - i\Gamma_{2,1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} T$

$= \begin{bmatrix} q & E_{1,2} + i\Gamma_{1,2} \\ E_{2,1} + i\Gamma_{2,1} & q^* \end{bmatrix} - \begin{bmatrix} E_{1,2} - i\Gamma_{1,2} & q \\ q^* & E_{2,1} - i\Gamma_{2,1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} T$

what about this term? - In what order do I operate?

after acting on each other

$\begin{bmatrix} E_{1,2} + i\Gamma_{1,2} & q^* \\ q & E_{2,1} - i\Gamma_{2,1} \end{bmatrix} T$

do the operators stay around or can I erase them?

Normally the operators do disappear - as long as it acted on everything to the right of it.
But since the $\mathcal{T}$ operator is an antilinear, I am less sure.

Week 5 (29/03/21 – 04/04/21)

Aims:

1. Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)
2. Reproduce Fig 1 in Bender
3. Simulate a RF-CAP

Tasks:

1. Meeting with J&M (Wednesday 3:00pm?)
   - ~~Ask J&M for feedback on my progress report~~
   - ~~Ask them about my issues understanding 2x2 model in Gao et al.~~
2. ~~SUBMIT PROGRESS REPORT~~
3. investigating the further papers on the E-P quantum billiard, or EPs done by Ostrovskaya's group

29/03/21

$$[\mathcal{PT}, H] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{T} \begin{bmatrix} E_{1,1} - i\Gamma_{1,1} & q \\ q^* & E_{1,1} - i\Gamma_{1,1} \end{bmatrix} - \begin{bmatrix} E_{1,1} - i\Gamma_{1,1} & q \\ q^* & E_{1,1} - i\Gamma_{1,1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{T}$$

$$= \begin{bmatrix} q & E_{1,1} + i\Gamma_{1,1} \\ E_{1,1} + i\Gamma_{1,1} & q^* \end{bmatrix} - \begin{bmatrix} q & E_{1,1} - i\Gamma_{1,1} \\ E_{1,1} - i\Gamma_{1,1} & q^* \end{bmatrix} \mathcal{T}$$

?

Assuming $\mathcal{T}$ operates from the right as it does from the left

$$[\mathcal{PT}, H] = \begin{bmatrix} q & E_{1,1} + i\Gamma_{1,1} \\ E_{1,1} + i\Gamma_{1,1} & q^* \end{bmatrix} - \begin{bmatrix} q^* & E_{1,1} + i\Gamma_{1,1} \\ E_{1,1} + i\Gamma_{1,1} & q \end{bmatrix}$$

$$= \begin{bmatrix} q - q^* & E_{1,1} + i\Gamma_{1,1} - E_{1,1} + i\Gamma_{1,1} \\ E_{1,1} + i\Gamma_{1,1} - E_{1,1} + i\Gamma_{1,1} & q^* - q \end{bmatrix} \neq 0$$

Note: there are edge cases when the commutator could be zero.
I know that if q is real, but i will also need $\tilde{E}_{1,2} = \tilde{E}_{2,1}$ and $\Gamma_{1,2} = -\Gamma_{2,1}$
I suspect that this is not applicable in general.

I can try a different definition: A Hamiltonian H is $\mathcal{PT}$-symmetric if $H^{\mathcal{PT}} = (\mathcal{PT})H(\mathcal{PT}) = H$

where: $\mathcal{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $\mathcal{T} \rightarrow$ complex inversion

$$\therefore (\mathcal{PT})H(\mathcal{PT}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{T} \begin{bmatrix} E_{1,1} - i\Gamma_{1,1} & q \\ q^* & E_{1,1} - i\Gamma_{1,1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{T}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{T} \begin{bmatrix} q^* & E_{1,1} - i\Gamma_{1,1} \\ E_{1,1} - i\Gamma_{1,1} & q \end{bmatrix} \mathcal{T}$$

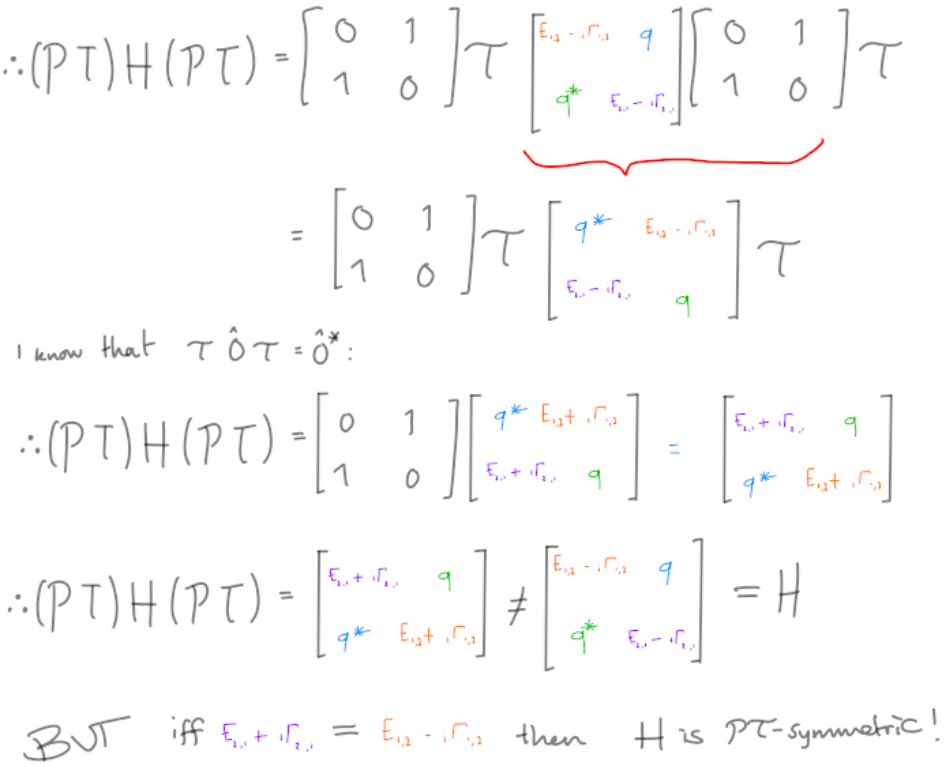
I know that $\mathcal{T}\hat{O}\mathcal{T} = \hat{O}^*$:

$$\therefore (\mathcal{PT})H(\mathcal{PT}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} q & E_{1,1} + i\Gamma_{1,1} \\ E_{1,1} + i\Gamma_{1,1} & q^* \end{bmatrix} = \begin{bmatrix} E_{1,1} + i\Gamma_{1,1} & q^* \\ q & E_{1,1} + i\Gamma_{1,1} \end{bmatrix}$$

$$\therefore (\mathcal{PT})H(\mathcal{PT}) = \begin{bmatrix} E_{1,1} + i\Gamma_{1,1} & q^* \\ q & E_{1,1} + i\Gamma_{1,1} \end{bmatrix} \neq \begin{bmatrix} E_{1,1} - i\Gamma_{1,1} & q \\ q^* & E_{1,1} - i\Gamma_{1,1} \end{bmatrix} = H \quad \text{in general}$$

H does not appear to be PT-symmetric in general. (?)

30/03/21
A colleague (Olivier) from my group found an error in my work. He explained that:
I forgot to conjugate q→ q* and q*→q when taking the complex conjugation (between the 2nd and 3rd lines).
He also mentioned that denoting the complex diagonal elements of the original matrix as **A11 and A22**, then the Hamiltonian is $\mathcal{PT}$-symmetric **iff A22*=A11**. Hence, A11 and A22 should have the same real parts, and exactly opposite imaginary parts. Essentially, the same conditions as I suspected earlier apply.



This is basically the same result obtained from the commutator analysis I did earlier.
I really need to understand the relations between the coupling strengths: $\Omega \leftrightarrow q$

31/03/21
MEETING Day!

2. Find eigenvalues of H

I am Diagonalizing H using mathematica, first I write $E = \lambda$(since E symbol representing the base of the natural logarithm)
Then I find the eigenvectors of the matrix, and diagonalize it!

```
In[1]:= mat = {{λ1 - I Γ1, q}, {q*, λ2 - I Γ2}};
        mat // MatrixForm
```

Out[2]//MatrixForm=

$$\begin{pmatrix} -i\,\Gamma_1 + \lambda_1 & q \\ q^* & -i\,\Gamma_2 + \lambda_2 \end{pmatrix}$$

Diagonalizing H:

```
In[3]:= vecs = Eigenvectors[mat];
        Inverse[Transpose[vecs]].mat.Transpose[vecs] // Chop
```

But the output is… not very pretty

Out[4]=

$$\left\{\left\{-\frac{q\,q^*}{\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \frac{(-i\Gamma_2+\lambda_2)\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2-\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)}{2\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \frac{1}{2\,q^*}\right.$$
$$\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2+\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)\left(-\frac{(-i\Gamma_1+\lambda_1)\,q^*}{\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \right.$$
$$\left.\left.\frac{q^*\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2-\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)}{2\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}}\right)\right\},$$
$$\left\{-\frac{q\,q^*}{\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \frac{(-i\Gamma_2+\lambda_2)\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2-\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)}{2\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \frac{1}{2\,q^*}\right.$$
$$\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2+\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)\left(-\frac{(-i\Gamma_1+\lambda_1)\,q^*}{\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}} - \right.$$
$$\left.\left.\frac{q^*\left(i\Gamma_1-i\Gamma_2-\lambda_1+\lambda_2-\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}\right)}{2\sqrt{-\Gamma_1^2+2\Gamma_1\Gamma_2-\Gamma_2^2-2i\Gamma_1\lambda_1+2i\Gamma_2\lambda_1+\lambda_1^2+2i\Gamma_1\lambda_2-2i\Gamma_2\lambda_2-2\lambda_1\lambda_2+\lambda_2^2+4q\,q^*}}\right)\right\}$$

Bender proceeds to analyse the parametric regions of the space from the eigenvalues in his example… But mine looks a bit too tricky...

I am a bit stuck on this part now...!

During our meeting I explained to J&M what I have understood so far and what I have not.
They both agree on my interpretation of the gain/decay rates in the main diagonal. Seems like $\Gamma_{n,n'}$ should be equal in magnitude but opposite in sign between the two coupled modes. But we were not able to find a reason on why I have a different sign for the imaginary part to the model equation (1) :

$$\tilde{E}_{n,n'} = E_{n,n'} - i\,\Gamma_{n,n'}$$

Neither of us yet understand the relation between Ωand *q* properly.
In fact all three of us feel a bit confused by the definition in the paper for q (supposedly determined by the degree of spatial overlap between the two modes,)

$$q \propto \int \varphi_n^*(\boldsymbol{r})\varphi_{n'}(\boldsymbol{r})\,\mathrm{d}^2\boldsymbol{r}$$

This definition is confusing because as I already stated earlier the basis vectors $|\varphi_{n'}\rangle$, $|\varphi_n\rangle$ are supposed to be orthogonal, which applies to their conjugates also. Meera suggested I find more information on this result by investigating the following papers on this subject done by Ostrovskaya's group. Maybe they will have further information that I can find useful to figure out this issue.

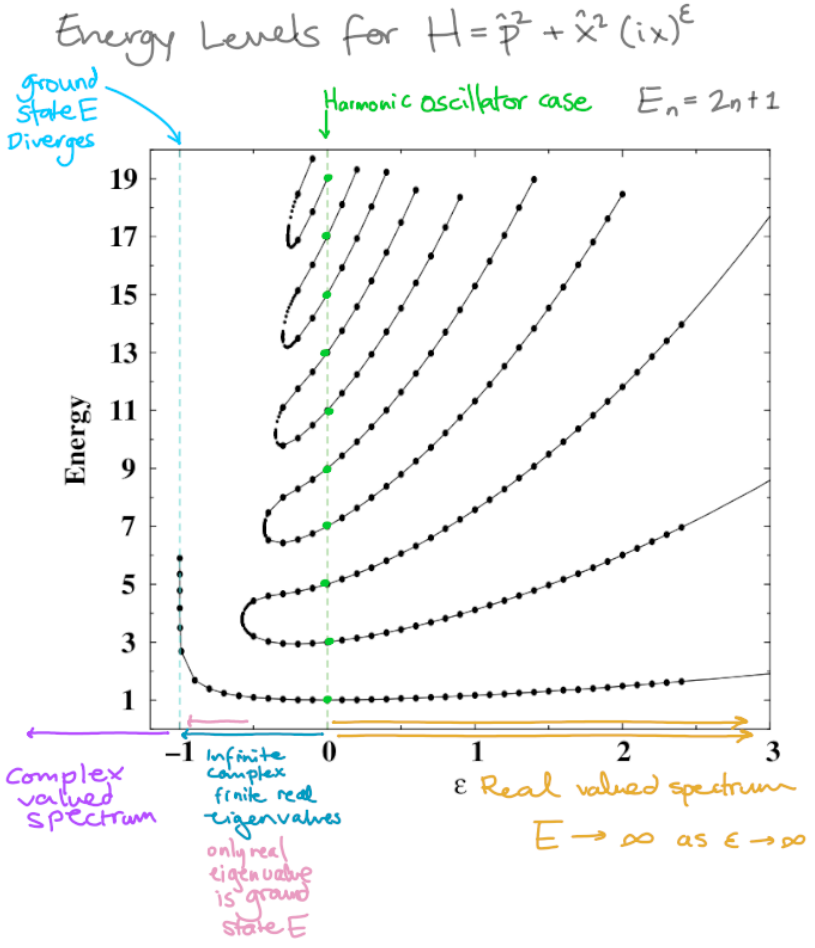
Jesper was wondering if the $\mathcal{T}$operator ever yields any non-trivial results. As defined it seems like a very simple operator. I told Jesper that from what I have read I think it doesn't have any sort of particularly interesting behaviour. But I am unsure.

Unfortunately we weren't able to get to the eigenvalues part of my work, so I could not show them how unaesthetic they appear.

~Now, to something else entirely...~

Reproducing Figure 1 in Bender

The general parametric family of $\mathcal{PT}$-symmetric Hamiltonians $H = \hat{p}^2 + \hat{x}^2\,(i\,\hat{x})^{\epsilon}$ as a function of the real parameter ϵ.
When $\epsilon \geq 0$, the $\mathcal{PT}$-symmetry is unbroken, but when $\epsilon < 0$the $\mathcal{PT}$-symmetry is broken.



The three regions:

1. $\epsilon \geq 0$, the spectrum is real and positive and the energy levels rise with increasing ϵ.
2. $-1 \leq \epsilon \leq 0$, there is a finite number of real positive eigenvalues and an infinite number of complex-conjugate pairs of eigenvalues (as ϵ decreases from 0 to −1, the number of real eigenvalues decreases).
3. For $\epsilon \leq -1$ there are no real eigenvalues.

Note:
$\epsilon = 0$, corresponds to the harmonic oscillator,
$\epsilon \leq -0.57793$The only real eigenvalue is the ground-state energy.
As $\varepsilon \to -1^+$, the ground-state energy diverges.

The principle behind this image is that we can think of the family of non-Hermitian Hamiltonians $H = \hat{p}^2 + \hat{x}^2(i\hat{x})^\epsilon$ as complex extensions of the 1D harmonic oscillator (which is Hermitian and $\mathcal{PT}$-symmetric) by introducing a real parameter ϵ in such a way that as ϵ increases from 0 the Hamiltonian is no longer Hermitian but its $\mathcal{PT}$-symmetry is maintained. This can be done with any of the Hermitian Hamiltonians $H = \hat{p}^2 + \hat{x}^{2N}$ where $N \in \mathbb{N}$ (Bender 953).
Bender claims that boundary conditions imposed on the eigenfunctions are the key enforcers of $\mathcal{PT}$-symmetry and hence are responsible for the "reality" of the energy spectrum of a Hamiltonian (Bender 959).

It is impossible to solve $-\psi'' + x^2(ix)^\epsilon\psi = E\psi$ analytically and in closed form in the cases $\epsilon = 0, \epsilon \to \infty$
Thus it is necessary to use approximate analytic or numerical methods

02/04/21

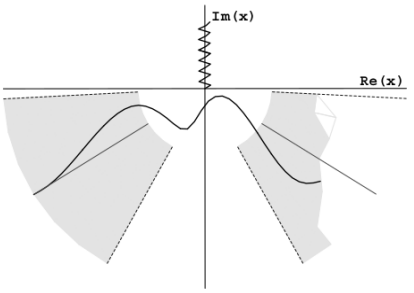
WKB integral techniques to calculate eigenvalues

- WKB is a great approximation to the eigenvalues when $\epsilon > 0$.
- WKB must be performed on the complex plane.

The turning points $x_\pm$ are those roots of $E = x^2(ix)^\epsilon$ that analytically continue off the real axis as ϵ increases from 0.
These turning points

$$x_- = E^{\frac{1}{\epsilon+2}} e^{i\pi(\frac{3}{2}-\frac{1}{\epsilon+2})} \qquad x_+ = E^{\frac{1}{\epsilon+2}} e^{-i\pi(\frac{1}{2}-\frac{1}{\epsilon+2})}$$

Lie in the lower-half (upper-half) x-plane when $\epsilon > 0 (\epsilon < 0)$ of fig 2. In Bender



I don't actually understand these regions well enough though, why are there turning points?

Bender references the following paper in his work so I think I should read this paper to get a more detailed methodology on the WKB method and the Stokes wedges regions in the complex x-plane.

Complex WKB Analysis of a $\mathcal{PT}$ Symmetric Eigenvalue Problem

Mark Sorrell

Department of Mathematics and the Maxwell Institute for Mathematical Sciences
Heriot-Watt University, Edinburgh, EH14 4AS, UK

March 2007

Abstract

The spectra of a particular class of $\mathcal{PT}$ symmetric eigenvalue problems has previously been studied, and found to have an extremely rich structure. In this paper we present an explanation for these spectral properties in terms of quantisation conditions obtained from the complex WKB method. In particular, we consider the relation of the quantisation conditions to the reality and positivity properties of the eigenvalues. The methods are also used to examine further the pattern of eigenvalue degeneracies observed by Dorey *et al.* in [1,2].

Complex WKB (?)

The complex WKB method involves treating x as a complex variable, and calculating how the asymptotic form of the solutions change as they are traced around the complex plane (Sorrell 4).
For a 1D TISE:

$$-\psi'' + V(x)\psi = E\psi$$

with asymptotic solutions for $|x| \to \infty$

$$\psi_\pm \sim \frac{1}{P(x)^{\frac{1}{4}}} \exp\left(\pm i \int_{x_0}^{x} dt\, (P(t))^{\frac{1}{2}}\right) \quad \text{where } P(x) = E - V(x)$$

The solutions are valid in some region of the complex plane around $x_0 (P(x) = 0$ ie. a turning point)

The integral in the exponent is in general complex valued. The real exponential parts of the solutions will either be growing (dominant) or decaying (subdominant). IF the integral is purely real then the solutions will be purely oscillatory (no domination)(Sorrell 4).

Stokes and Anti-stokes lines

The curves where $\Im m(\int_{x_0}^{x} P(t)^{\frac{1}{2}} dt) = 0$ are known as anti-Stokes lines. Anti-Stokes lines mark the borders between sectors of dominant/subdominant behaviour; in a given sector one solution will be dominant, but on crossing an **anti-Stokes line** to the next sector the dominance behaviour reverses and the solution will be subdominant in the new sector. We can also find the regions where $\Re e(\int_{x_0}^{x} P(t)^{\frac{1}{2}} dt) = 0$. These are known as Stokes lines, and are the regions where the solutions are most dominant/subdominant. Upon crossing a **Stokes line**, the coefficient of the subdominant solution changes by an amount proportional to the coefficient of the dominant term. Knowing the position of Stokes and anti-Stokes lines is crucial in applying **complex WKB methods**(Sorrell 4).

Crossing a Stokes line: new_subdominant_coeff = old_subdominant_coeff + T ×old_dominant_coeff
T is the Stokes multiplier and it depends on the nature of the turning point where the Stokes line originates from.

If: the Stokes line is emanating from a simple turning point $T = i$,
Else: the Stokes line is emanating from a double zero $T = \sqrt{2i}$ **(to first-order)**

*Note: new_dominant_coeff = old_dominant_coeff

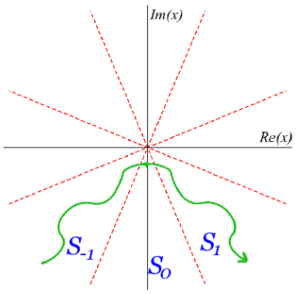
To connect solutions defined in terms of different turning points, x_1 and x_2, use:

$$\exp\left(\pm i \int_{x_1}^{x} (P(t))^{\frac{1}{2}} dt\right) = \exp\left(\pm i \int_{x_1}^{x_2} (P(t))^{\frac{1}{2}} dt\right) \exp\left(\pm i \int_{x_2}^{x} (P(t))^{\frac{1}{2}} dt\right).$$

Note: it is chosen to introduce the discontinuous change in the subdominant coefficient at the Stokes line, so that this discontinuity is small compared to the error in the WKB approximation(Sorrell 5).

WKB Quantization conditions for eigenvalue problems

By starting with a WKB solution that is purely subdominant (decaying) in the Stokes sector S_{-1}



Tracing the solution around the complex plane, in an anti-clockwise direction.
As Stokes lines are crossed new subdominant exponentials appear. The new subdominant contributions will have a coefficient of i (from Stokes line from a simple turning point) or $\sqrt{2i}$(Stokes line from a second order zero) multiplied by the coefficient of the dominant exponential at that Stokes line leads to a wave function in the Stokes sector S_1 that has both a subdominant and dominant components(Sorrell 5).

Different Stokes' structures are obtained depending on whether the double zeros occur within, outside of, or directly in line with the other turning points.

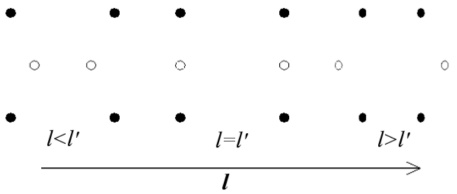


Fig 5. In Sorrell: Relative positions of turning points.
The double zeros (white dots) with respect to single zeros (black dots).

For a given value of the parameter α, the value of l required for the zeros that coalesce to lineup with the others, (denoted l'), is given by

$$l' = -\frac{1}{2} + \frac{\sqrt{(1+\alpha^2)}}{2}$$

The organisation of the different possible structures of the stokes lines in the plane due to the **sequence of turning point configurations** depends on the value of l relative to l' for a given α(Sorrell 10).
The (α, l) plane can be divided up into sectors according to which sequence of turning point arrangements is valid there.
Note: A sequence is obtained because the pattern of zeros obtained also depends on energy (Sorrell 10).

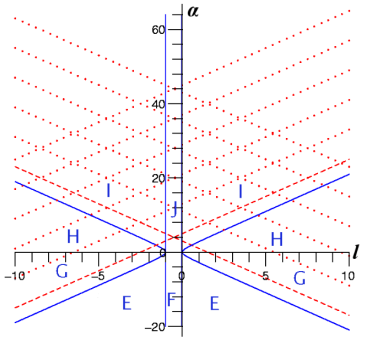


Figure 6: **Regions of the α - l plane** . *The solid lines show how the plane is split up in terms of different Stokes structures occurring. The dashed lines indicate the boundaries from [1, 2].*

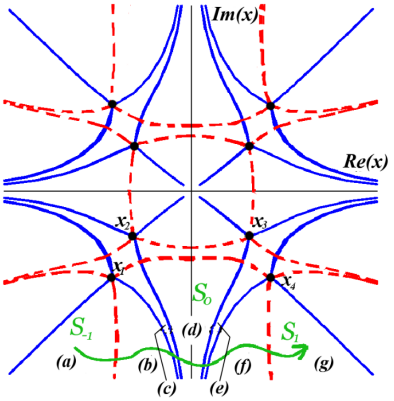


Figure 7: **Stokes structure,** $\alpha = 3$, $l = 0.5$, $E = 1$ *Stokes lines are shown as solid lines and anti-Stokes lines are the broken lines.*

I feel like I don't actually understand the concept of the Stokes structures in the complex plane and I don't actually have a proper plan to replicate Fig 1 in Bender yet.

04/04/2021

Boundary conditions for the Schrödinger eigenvalue problem (Bender 956)

To solve the Schrödinger eigenvalue problem we write the Schrödinger equation in coordinate space. The question is, what are the BCs??
In coordinate space the operators $x \rightarrow x$ and $p \rightarrow -i\frac{d}{dx}$, we want to treat the variable x as complex.

As we know the Schrödinger eigenvalue problem for $H = \hat{p}^2 + \hat{x}^2 (i\hat{x})^\epsilon$ takes the form $-\psi'' + x^2(ix)^\epsilon\psi = E\psi$.
We cannot solve this equation for an arbitrary ε (it is impossible to solve the differential equation eigenvalue problem analytically and in closed form except in two special cases, namely, for ε = 0 (the harmonic oscillator) and for $\varepsilon \rightarrow \infty$).
Therefore we use the WKB approximation to find asymptotic solutions:

WKB approximation:
for ODE of the form:

$$-\psi'' + V(x)\psi = E\psi \quad \text{where } V \rightarrow \infty \text{ as } |x| \rightarrow \infty$$

assume $\psi \sim \exp\left[\pm \int^x ds\sqrt{V(s)}\right]$

wavefunction moving in space

The leading order WKB phase-integral quantization condition

$$(n+\tfrac{1}{2})\pi = \int_{x_-}^{x_+} dx \sqrt{E - x^2(ix)^\epsilon}$$

AKA how much phase is gained by traversing from $x_- \rightarrow x_+$
if $\epsilon < 0$ WKB fails ∵ Path is not continuous between $x_- \rightarrow x_+$
when $\epsilon \geq 0$ the phase integral contour is deformed to

$$(n+\tfrac{1}{2})\pi = 2\sin\left(\frac{\pi}{\epsilon+2}\right) E^{\frac{1}{\epsilon+1}+\frac{1}{2}} \int_0^1 ds \sqrt{1 - s^{\epsilon+2}}$$

Therefore an approximation to the eigenvalues in fig 1. is:

$$\therefore E_n \sim \left(\frac{\Gamma\left(\frac{3}{2} + \frac{1}{\epsilon+2}\right)\sqrt{\pi}\,(n+\frac{1}{2})}{\sin\left(\frac{\pi}{\epsilon+2}\right)\Gamma\left(1+\frac{1}{\epsilon+2}\right)} \right)^{\frac{2\epsilon+4}{\epsilon+4}}$$

When ε > 0 this path lies entirely in the lower-half x plane, and when ε = 0 (the case of the harmonic oscillator) the path lies on the real axis. However, when ε < 0 the path lies in the upper-half x plane and crosses the cut on the positive imaginary-x axis. In this case there is no continuous path joining the turning points. Hence, **WKB fails when ε< 0.**

According to Bender, the WKB formula gives a very accurate approximation to the eigenvalues plotted in figure 1. and it shows, at least in the WKB approximation, that the energy eigenvalues of $H = \hat{p}^2 + \hat{x}^2 (i\hat{x})^\epsilon$ are real and positive (Bender 960).

Table 1. Comparison of the exact eigenvalues (obtained with Runge–Kutta) and the WKB result in (34).

ϵ	n	E_{exact}	E_{WKB}	ϵ	n	E_{exact}	E_{WKB}
1	0	1.156 267 072	1.0943	2	0	1.477 149 753	1.3765
	1	4.109 228 752	4.0895		1	6.003 386 082	5.9558
	2	7.562 273 854	7.5489		2	11.802 433 593	11.7690
	3	11.314 421 818	11.3043		3	18.458 818 694	18.4321
	4	15.291 553 748	15.2832		4	25.791 792 423	25.7692
	5	19.451 529 125	19.4444		5	33.694 279 298	33.6746
	6	23.766 740 439	23.7606		6	42.093 814 569	42.0761
	7	28.217 524 934	28.2120		7	50.937 278 826	50.9214
	8	32.789 082 922	32.7841		8	60.185 767 651	60.1696
	9	37.469 824 697	37.4653		9	69.795 703 031	69.7884

I don't really follow the method further outlined involving complex integration of contours within the stokes regions.
I will therefore go to the next section, explaining the numerical calculation.

Numerical calculation of eigenvalues

I require

the leading order WKB phase-integral quantization condition

$$(n+\tfrac{1}{2})\pi = \int_{x_-}^{x_+} dx \sqrt{E - x^2(ix)^\epsilon}$$

Where the **turning points**

$$x_- = E^{\frac{1}{\epsilon+2}} e^{i\pi(\frac{3}{2}-\frac{1}{\epsilon+2})} \qquad x_+ = E^{\frac{1}{\epsilon+2}} e^{-i\pi(\frac{1}{2}-\frac{1}{\epsilon+2})}$$

Here I have written the basic set up of what I think will give me the values of the eigen energies based on the WKB methodology
Where the LHS is the quantization condition I've written in red above and the RHS is the WKB integral that defines the energy

Bender1_.py

```
# PHS3350
# Week 5 - Energy levels of a family of non-Hermitian Hamiltonians
# Ana Fabela Hinojosa, 04/04/2021

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.optimize import fsolve

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

def integrand(x, E, ϵ):
    return np.sqrt(E - x**2*(1j*x)**ϵ)

def RHS(E, ϵ):
    tp_minus = E**(1/(ϵ+2)) * np.exp(1j * np.pi * (3/2 - (1/(ϵ+2))))
    tp_plus = E**(1/(ϵ+2)) * np.exp(-1j * np.pi * (3/2 - (1/(ϵ+2))))
    return quad(integrand, tp_minus, tp_plus, args=(E, ϵ))

def LHS(n):
    return (n +1/2) * np.pi

def error(E, ϵ, n):
    return RHS(E, ϵ) - LHS(n)

for n in range(15):
    for ϵ in np.linspace(-1, 3, 512):
        fsolve(error, 0, args=(ϵ, n))


# If fsolve was less smart...
# for n in range(15):
#     for ϵ in np.linspace(-1, 3, 512):
#         def err(E):
#             return error(E, ϵ, n)

```

```
ana@ana-XPS-13-9343: ~/Desktop/proj/code
[ana:code]$ python3 Bender1_.py
Traceback (most recent call last):
  File "Bender1_.py", line 29, in <module>
    fsolve(error, 0, args=(ϵ, n))
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/optimize/minpack.py", line 147, in fsolve
    res = _root_hybr(func, x0, args, jac=fprime, **options)
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/optimize/minpack.py", line 213, in _root_hybr
    shape, dtype = _check_func('fsolve', 'func', func, x0, args, n, (n,))
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/optimize/minpack.py", line 26, in _check_func
    res = atleast_1d(thefunc(*((x0[:numinputs],) + args)))
  File "Bender1_.py", line 25, in error
    return RHS(E, ϵ) - LHS(n)
  File "Bender1_.py", line 19, in RHS
    return quad(integrand, tp_minus, tp_plus, args=(E, ϵ))
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 341, in quad
    retval = _quad(func, a, b, args, full_output, epsabs, epsrel, limit,
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 453, in _quad
    return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
TypeError: can't convert complex to float
[ana:code]$
```

I think that I am getting an error because I am using the complex turning points as the limits in the integral. I am performing a complex integral.
Maybe I need a change of variables so I can translate those points into the real line.

Important questions:

- is $q \propto \int \varphi_n^*(\boldsymbol{r})\varphi_{n'}(\boldsymbol{r})\, \mathrm{d}^2\boldsymbol{r}$ a definition that makes sense?

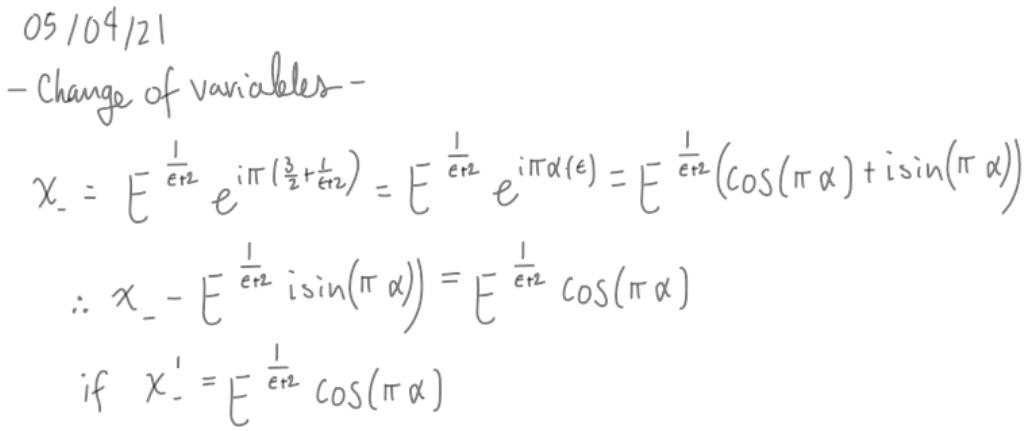
MID-SEM break (05/04/21 – 11/04/21)

Aims:

1. Reproduce Fig 1 in Bender
2. Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)
3. Simulate a RF-CAP

Tasks:

1. F~~ix complex integral by change of variables~~
2. investigating the further papers on the E-P quantum billiard, or EPs done by Ostrovskaya's group (?)



After this kind change of variables for both my integration limits I still obtain the same TypeError: can't convert complex to float

Possible approach

- Complex integral?
  - Solving real and imaginary part separately

```
def complex_quad(func, a, b, **kwargs):
    def real_func(*args):
        return np.real(func(*args))
    def imag_func(*args):
        return np.imag(func(*args))
    real_integral = quad(real_func, a, b, **kwargs)
    imag_integral = quad(imag_func, a, b, **kwargs)
    print(real_integral[0] + 1j*imag_integral[0])
    return real_integral[0] + 1j*imag_integral[0]
```

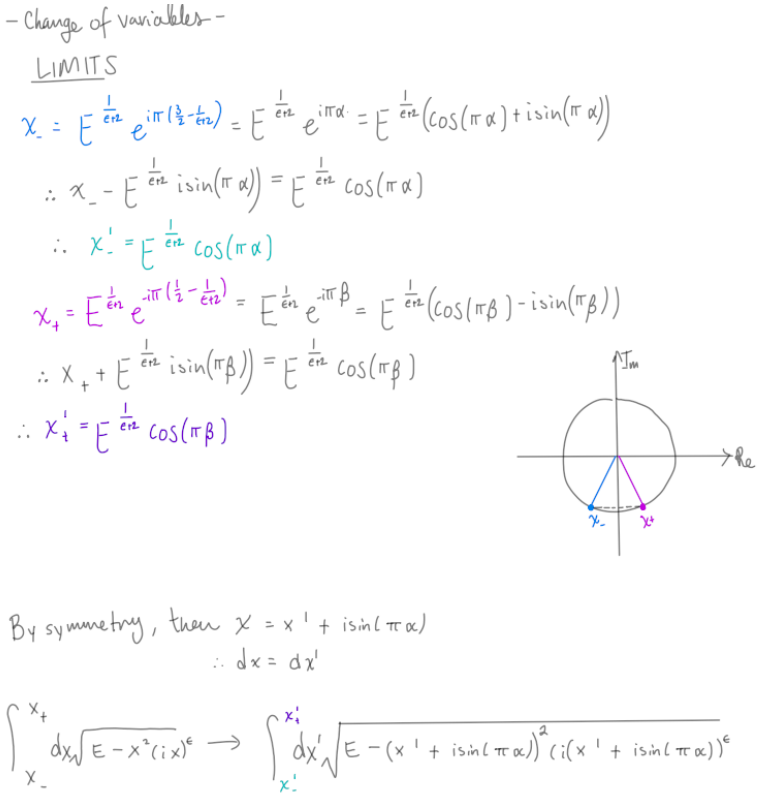
- Calling fsolve twice?
  - real and imaginary part respectively (also)

```
def complex_fsolve(func, E0, **kwargs):
    def real_func(*args):
        return np.real(func(*args))
    def imag_func(*args):
        return np.imag(func(*args))
    real_root = fsolve(real_func, E0, **kwargs)
    imag_root = fsolve(imag_func, E0, **kwargs)
    print(real_root, imag_root)
    if np.isclose(real_root, imag_root):
        print(f"found a root: E = {real_root}")
        return real_root
    else:
        print("no root for you")
```

Test call:
complex_fsolve(error, E0=47, args=(3, 200))

```
[ana:code]$ python3 Bender1_.py
(17.407702450933172-1.9663836681374992e-17j)
(17.407702450933172-1.9663836681374992e-17j)
(17.407702450933172-1.9663836681374992e-17j)
(17.407702632509654-3.5739615535758425e-17j)
(273.90991171953607-5.417984203366521e-16j)
(499.7383910957591-1.5540236785914897e-15j)
(610.8588044751805-6.633073671783454e-16j)
(629.004912638415-7.863695762417821e-16j)
(629.8835663439817-9.02818161320961e-16j)
(629.8893253110411-2.593587423133634e-15j)
(629.8893270447501-2.1680332615093876e-15j)
(17.407702450933172-1.9663836681374992e-17j)
(17.407702450933172-1.9663836681374992e-17j)
(17.407702450933172-1.9663836681374992e-17j)
(17.407702632509654-3.5739615535758425e-17j)
(17.407702228829447-3.8506676173148804e-17j)
[7916.96989632] [47.]
no root for you
[ana:code]$
```

It works (?)… Also the calculated output for the imaginary part of the RHS is very small! Therefore the code assumes it is negligible.
Chris noticed that I haven't finished my change of variables completely OH NO!
I haven't changed my integrand!!



Corrected INTEGRAND

```python
def integrand(x_prime, E, ϵ):
    # OG integrand: np.sqrt(E - x**2 * (1j * x)**ϵ)
    # Change of variables
    α = 3/2 - 1/(ϵ +2)
    x = x_prime + 1j * np.sin(np.pi * α)
    return np.sqrt(E - x**2 * (1j * x)**ϵ)
```

OUTPUT:

```
[ana:code]$ python3 Bender1_.py
(1.0527491023071227-1.6894383256689398e-18j)
(1.0527491023071227-1.6894383256689398e-18j)
(1.0527491023071227-1.6894383256689398e-18j)
(1.052749113280112-1.8836807897815615e-17j)
(3.9311950217475853+4.710114770230738e-17j)
(4.5615012971840905+8.697603454171054e-17j)
(4.707207846199189+3.358565876381156e-17j)
(4.712355659800096+7.68091887805674e-17j)
(4.71238897306821+5.2676735070455087e-17j)
(4.712388980384681+4.647671872674304e-17j)
(1.0527491023071227-1.6894383256689398e-18j)
(1.0527491023071227-1.6894383256689398e-18j)
(1.0527491023071227-1.6894383256689398e-18j)
(1.052749113280112-1.8836807897815615e-17j)
(1.052749101225963+2.478709464225006e-17j)
found a root: E = [7.65292902]
```

Imaginary part resulting from the complex integration remains negligible.
Which it's probably correct.

06/04/21

TESTS

I want to compare the obtain values for a few cases displayed in table 1 in Bender

Table 1. Comparison of the exact eigenvalues (obtained with Runge–Kutta) and the WKB result in (34).

ϵ	n	E_{exact}	E_{WKB}	ϵ	n	E_{exact}	E_{WKB}
1	0	1.156 267 072	1.0943	2	0	1.477 149 753	1.3765
	1	4.109 228 752	4.0895		1	6.003 386 082	5.9558
	2	7.562 273 854	7.5489		2	11.802 433 593	11.7690
	3	11.314 421 818	11.3043		3	18.458 818 694	18.4321
	4	15.291 553 748	15.2832		4	25.791 792 423	25.7692
	5	19.451 529 125	19.4444		5	33.694 279 298	33.6746
	6	23.766 740 439	23.7606		6	42.093 814 569	42.0761
	7	28.217 524 934	28.2120		7	50.937 278 826	50.9214
	8	32.789 082 922	32.7841		8	60.185 767 651	60.1696
	9	37.469 824 697	37.4653		9	69.795 703 031	69.7884

- IF $(n, \epsilon) = (0, 1)$
  THEN my code output is:

```
(n, ϵ) = (0, 1)
found a root: E = 3.0000
```

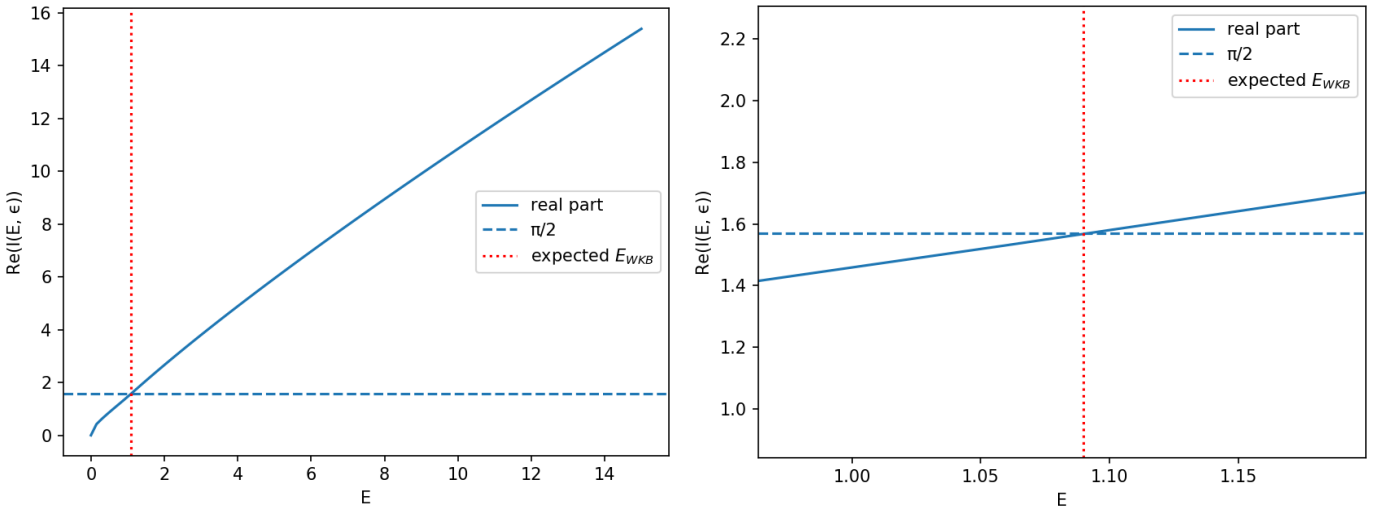
Which is not correct according to the table.

1. I have checked my integrand function and it seems like I have done my change of variables correctly.
2. I have checked my RHS function and the limits I defined for the integral seem to be correct also.

I want to plot the real part of my integrand function as a function of E.
Maybe I can see where it crosses zero. Since square roots have 2 possible solutions, maybe the plot of the integrand will show if there are any discontinuities in the function from random choosings by numpy between the 2 possible solutions that it displays to me.

For this test I created a linspace of Es and used this to evaluate RHS(E, ϵ=1)
I stored the result of each integration done by RHS in a list and then plotted this vs the E linspace. Since the LHS(0) = π/2

```
##TEST
Is = []
for E in np.linspace(0, 15, 100):
    I = RHS(E, 1)
    Is.append(I)

E = np.linspace(0, 15, 100)
plt.plot(E, np.real(Is), label="real part")
plt.axhline(np.pi *0.5, linestyle='--', label="π/2")
plt.axvline(1.09, linestyle=':', color='r', label=r"expected $E_{WKB}$")
plt.legend()
plt.xlabel("E")
plt.ylabel("Re(I(E, ϵ))")

plt.show()
```



I found the correct approximation to the energy value given the n and ϵparameters.
So, both my change of variables and integration function work correctly.
Then why do I obtain the wrong root when I use fsolve on error?

BUG SOLVED: I was feeding the parameters n, ϵ in the wrong order to fsolve!
This is the correct output:

```
(ϵ, n) = (1, 0)
found a root: E = 1.0925
```

It is less accurate than the WKB approximation by 0.1% :

```
number of σ away is WKB from exact result: 0.054
number of σ away am I from exact result: 0.055
```

I am going to proceed with an iterative approach to see if I can find close values for all $\epsilon = 1$ and all n in table table 1 in Bender.

```
# iterative approach for ϵ = 1 case:
Energies = []
for n in range(10):
  E = complex_fsolve(error, 1.1563, args=(1, n))
  Energies.append(E)
```

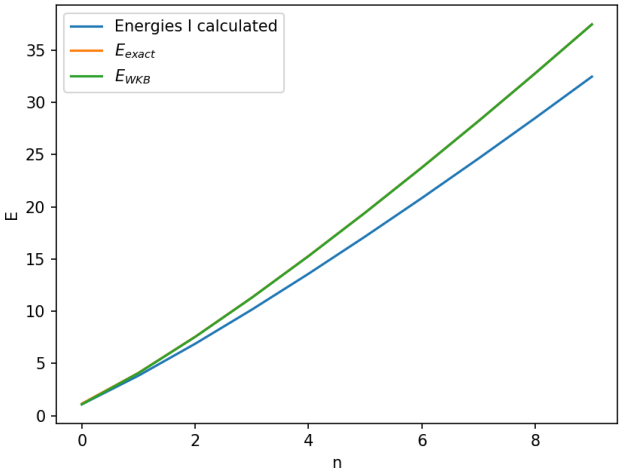
$\epsilon = 1$

```
n = 0, E = 1.0925
n = 1, E = 3.8381
n = 2, E = 6.8935
n = 3, E = 10.1611
n = 4, E = 13.5929
n = 5, E = 17.1600
n = 6, E = 20.8431
n = 7, E = 24.6283
n = 8, E = 28.5047
n = 9, E = 32.4640
```

Table 1. Comparison of the exact eigenvalues (obtained with Runge–Kutta) and the WKB result in (34).

ϵ	n	E_{exact}	E_{WKB}	ϵ	n	E_{exact}	E_{WKB}
1	0	1.156 267 072	1.0943	2	0	1.477 149 753	1.3765
	1	4.109 228 752	4.0895		1	6.003 386 082	5.9558
	2	7.562 273 854	7.5489		2	11.802 433 593	11.7690
	3	11.314 421 818	11.3043		3	18.458 818 694	18.4321
	4	15.291 553 748	15.2832		4	25.791 792 423	25.7692
	5	19.451 529 125	19.4444		5	33.694 279 298	33.6746
	6	23.766 740 439	23.7606		6	42.093 814 569	42.0761
	7	28.217 524 934	28.2120		7	50.937 278 826	50.9214
	8	32.789 082 922	32.7841		8	60.185 767 651	60.1696
	9	37.469 824 697	37.4653		9	69.795 703 031	69.7884

I obtain smaller values nearly each time. The **error grows** as the energy increases.

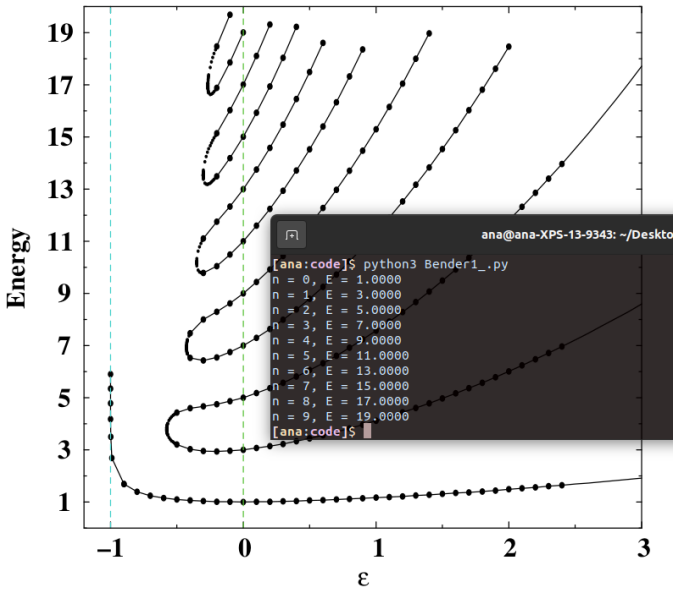


The exact energies reported in Bender are so close to the WKB energies that at this scale the error between them is invisible, unlike with my result.

I have tried increasing the accuracy of my integration function by decreasing the allowed error tolerance as the documentation describes but that has no impact on my outputs.

The eigenvalues of the harmonic oscillator case ie $\epsilon = 0$:

```
# iterative approach for ϵ = 0 case:
Energies = []
for n in range(10):
  E = complex_fsolve(error, 1.1563, args=(0, n))
  Energies.append(E)
  print(f"n = {n}, E = {E:.04f}")
```



I obtain the Harmonic oscillator eigenvalues exactly as the reported values in Bender.

I could do 2 more checks.

1. **FIRST TEST:** I want to try comparing Bender's analytic expression to my numerical results and see if I differ from his.

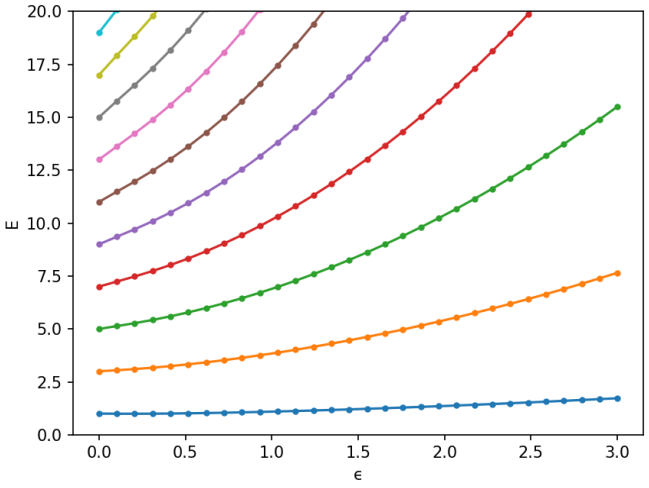
$$E_n \sim \left[\frac{\Gamma\left(\frac{3}{2} + \frac{1}{\epsilon+2}\right)\sqrt{\pi}\left(n+\frac{1}{2}\right)}{\sin\left(\frac{\pi}{\epsilon+2}\right)\Gamma\left(1+\frac{1}{\epsilon+2}\right)} \right]^{\frac{2\epsilon+4}{\epsilon+4}} \qquad (n \to \infty).$$

```python
def analytic_E(ϵ, n):
    top = gamma(3/2 + 1/(ϵ +2)) * np.sqrt(np.pi) * (n + 1/2)
    bottom = np.sin(np.pi / (ϵ + 2)) * gamma(1 + 1/(ϵ + 2))
    return (top/bottom)**((2 * ϵ + 4)/(ϵ + 2))
```

2. **IF FIRST TEST FAILS**: I would want to write a brute force integral to try and bypass any possible issues I might be having with scipy's quad() I could do this by simply summing over all integrand × dx'

Currently my code can plot the Hermitian region of the Hamiltonian family

```python
# ITERATIVE
Energies = []
for n in range(10):
    E_εs = []
    for ϵ in np.linspace(0, 3, 30):
        E_ϵ = complex_fsolve(error, 0, args=(ϵ, n))
        E_εs.append(E_ϵ)
        # print((E_εs)
    Energies.append(E_εs)

#PLOTING
for E_εs in Energies:
    # print(E_εs)
    ϵ = np.linspace(0, 3, 30)
    plt.plot(ϵ, E_εs, "o-", markersize=3)
plt.ylim(0, 20)
plt.xlabel("ϵ")
plt.ylabel("E")
plt.show()
```



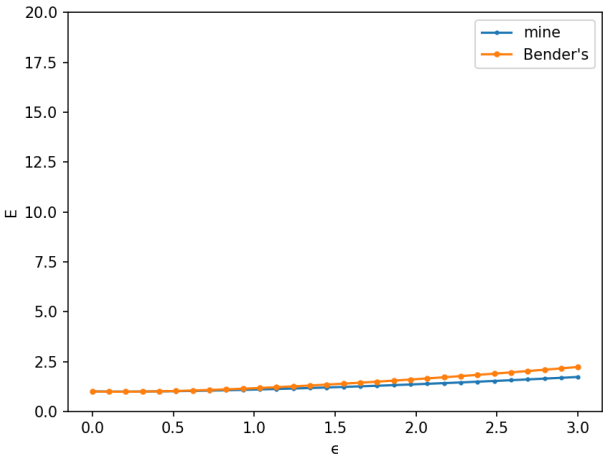
It is looking pretty good but again, i am not entirely sure of how precise it is. Given the comparative results vs WKB and RK.

I need to figure out how to do the broken symmetry sector of the spectrum ie. if $\epsilon < 0$

Test 1: comparison to the analytical solution in Bender

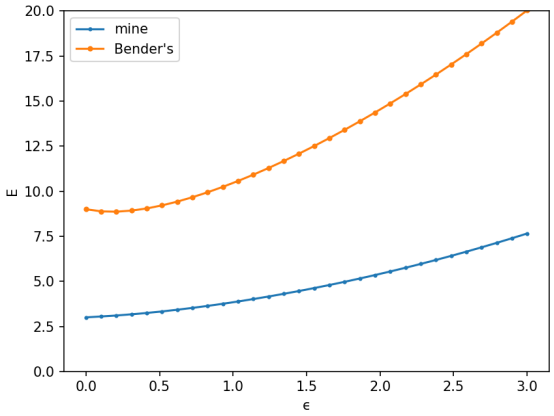
Here I am testing my method vs Bender's analytical result for many ϵbut with n=0:

```
## Single case Comparison to Bender's analytic result (ϵ, n=0)
E_ϵs_n0 = []
analytic_E_ϵs_n0 = []
for ϵ in np.linspace(0, 3, 30):
    # complex_fsolve(error, 1.1563, args=(1, 0))???
    E_ϵ_n0 = complex_fsolve(error, 0, args=(ϵ, 0))
    E_ϵs_n0.append(E_ϵ_n0)

    analytic_E_ϵ_n0 = analytic_E(ϵ, 0)
    analytic_E_ϵs_n0.append(analytic_E_ϵ_n0)

ϵ = np.linspace(0, 3, 30)
plt.plot(ϵ, E_ϵs_n0, "o-", markersize=2, label="mine")
plt.plot(ϵ, analytic_E_ϵs_n0, "o-", markersize=3, label="Bender's")
plt.legend()
plt.ylim(0, 20)
plt.xlabel("ϵ")
plt.ylabel("E")
plt.show()
```



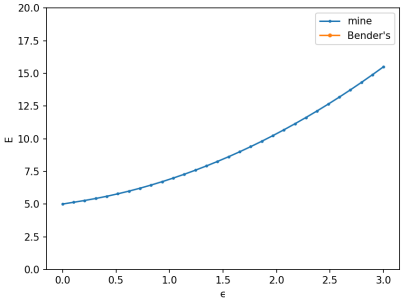
The plot looks the same if I make my initial guess for E in complex_fsolve E0 = 1.1563.

My current hypothesis: The issue might be in the precision of quad().



To obtain the figure on the left I simply made n=1.
It can be seen that the error is much larger than in the n=0 case.

The situation becomes weirder if I increase the n value to n=2 (as in the figure below). The solution reported by bender is simply not visible anymore.



This issue continues as n increases. Bender is off the charts! WHY?

Found a TYPO! After re reading Benders analytic solution I discovered an error in the power :D

```
def analytic_E(ϵ, n):
    top = gamma(3/2 + 1/(ϵ +2)) * np.sqrt(np.pi) * (n + 1/2)
    bottom = np.sin(np.pi / (ϵ + 2)) * gamma(1 + 1/(ϵ + 2))
    return (top/bottom)**((2 * ϵ + 4)/(ϵ + 2))
```

```
def analytic_E(ϵ, n):
    top = gamma(3/2 + 1/(ϵ +2)) * np.sqrt(np.pi) * (n + 1/2)
    bottom = np.sin(np.pi / (ϵ + 2)) * gamma(1 + 1/(ϵ + 2))
    return (top/bottom)**((2 * ϵ + 4)/(ϵ + 4))
```

09/04/21

To obtain this figure I set n = 2 once again and it can be seen that I obtain a better match with my result. But there is still divergence from the solution for large ϵ.
So I should try adjusting the accuracy of fsolve() and see if that makes a difference...Aaand it doesn't.
Below I have the numerical results compared to the Bender results.

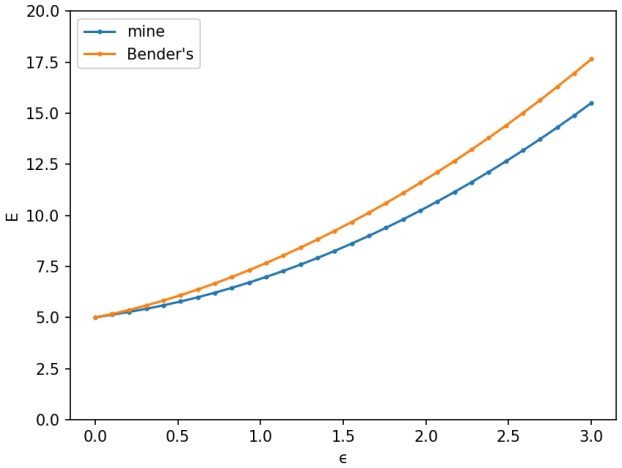


Table 1. Comparison of the exact eigenvalues (obtained with Runge–Kutta) and the in (34).

ϵ	n	E_{exact}	E_{WKB}	ϵ	n	E_{exact}	E_{WKB}
1	0	1.156 267 072	1.0943	2	0	1.477 149 753	1.3765
	1	4.109 228 752	4.0895		1	6.003 386 082	5.9558
	2	7.562 273 854	7.5489		2	11.802 433 593	11.7690
	3	11.314 421 818	11.3043		3	18.458 818 694	18.4321
	4	15.291 553 748	15.2832		4	25.791 792 423	25.7692
	5	19.451 529 125	19.4444		5	33.694 279 298	33.6746
	6	23.766 740 439	23.7606		6	42.093 814 569	42.0761
	7	28.217 524 934	28.2120		7	50.937 278 826	50.9214
	8	32.789 082 922	32.7841		8	60.185 767 651	60.1696
	9	37.469 824 697	37.4653		9	69.795 703 031	69.7884

```
ϵ = 1
n = 0, E = 1.0924666633369786
n = 1, E = 3.8381472820713776
n = 2, E = 6.893502957998225
n = 3, E = 10.161138186736617
n = 4, E = 13.592897290161986
n = 5, E = 17.160014670867742
n = 6, E = 20.84313934174705
n = 7, E = 24.628257773241426
n = 8, E = 28.504698313233455
n = 9, E = 32.464031524293596
```

```
ϵ = 2
n = 0, E = 1.3595997290545954
n = 1, E = 5.412898987396649
n = 2, E = 10.380459119287472
n = 3, E = 16.001123354110554
n = 4, E = 22.145909404687266
n = 5, E = 28.735585738303648
n = 6, E = 35.7155703787724
n = 7, E = 43.045533985981116
n = 8, E = 50.694228170883655
n = 9, E = 58.63659209200645
```

I will proceed with **test 2**
Using my obtained numerical results.

Test 2: The brute force integral.

```python
def brute_force(func, E, ϵ):
    def real_func(*args):
        return np.real(func(*args))
    # limits
    tp_minus_prime = E**(1/(ϵ+2)) * np.cos(np.pi * (3/2 - (1/(ϵ+2))))
    tp_plus_prime = E**(1/(ϵ+2)) * np.cos(np.pi * (1/2 - (1/(ϵ+2))))
    # domain & differential (infinitesimal)
    x_prime = np.linspace(tp_minus_prime, tp_plus_prime, 30)
    dx_prime = x_prime[1] - x_prime[0]
    return np.sum(real_func(x_prime, E, ϵ) * dx_prime)
```

Since I have obtained a list of energies from my method. I can use one of these E in this integral and see if it solves the quantization condition accurately
I am going to use the (arbitrarily chosen) energy value : ϵ = 1, n = 6, E = 20.84313934174705

Does the brute force integral solve the quantization condition?

```python
# FIND ENERGIES TO FEED INTO BRUTE INTEGRAL
Energies = []
# i = 0
for ϵ in range(3):
    # print(f"{ϵ = }")
    E_s = []
    for n in range(10):
        E = complex_fsolve(error, 1.1563, args=(ϵ, n))
        E_s.append(E)
        # print(f"{i = }, {n = }, {E = }")
        # i+=1
    Energies.append(E_s)

qc = (6 + 1/2) * np.pi
brute_integral = brute_force(integrand, Energies[1][6], 1)

print("\nCase: (ϵ, n) = (1,6)")
print(f"\n(n + 1/2)π = {qc:.04f}")
print(f"\nEnergy from complex_fsolve() E = {Energies[1][6]:.04f}")
print(f"\nEnergy from {brute_integral = :.04f}\n")
```

ϵ	n	E_{exact}	E_{WKB}
1	6	23.766 740 439	23.7606

```
Case: (ϵ, n) = (1,6)

(n + 1/2)π = 20.4204

Energy from complex_fsolve() E = 20.8431

Energy from brute_integral = 21.0520
```

The accuracy of the brute force integral increases as the infinitesimal differential decreases… this size is adjusted by changing the number of points in the linspace… I went for it and slowly increased the number from 30 →3000

```
Case: (ϵ, n) = (1,6)

(n + 1/2)π = 20.4204

Energy from complex_fsolve() E = 20.8431

Energy from brute_integral = 20.4265
```

Since I am using an energy I calculated using my complex_fsolve() function and passing it into the brute_ integral() it appears that my method does in fact satisfy the quantization condition when I have many very small integration steps!
This means that my functions: complex_quad() and complex_fsolve() are doing a good job!
The trend continues as I increase the number of integration steps in the region.
This tells me nothing about the discrepancy between my results and the reported results in Bender though. SAD reacc

Important questions:

- How can I figure out if the eigenvalues of Gao et al's 2x2 Hamiltonian are correct?
  What is next?
- What other test can I do in order to check my code vs Bender?
  Can Bender's results be wrong?
- When $\varepsilon < 0$ the path lies in the upper-half x plane and crosses the cut on the positive imaginary-x axis. In this case there is no continuous path joining the turning points. Hence, **WKB fails when $\varepsilon < 0$.**
  How do I obtain the Energy values on the broken symmetry sector of the spectrum?

Week 6 (12/04/21 – 18/04/21)

Aims:

1. Reproduce Fig 1 in Bender
2. Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)
3. Simulate a RF-CAP

Tasks:

1. ~~Ask for advice on discrepancy between mine and Bender results~~
   ~~-Reproducing fig 1 in Bender (TESTS)~~
2. ~~Show J&M my results for Gao et al.'s 2x2 Hamiltonian eigenvalues~~
   ~~-Mathematica code~~
3. ~~Email Dr. T Gao and Prof E Ostrovskaya and ask about the definition of *q*~~
4. **Understand how is it that complex eigenstates of NH systems are not orthogonal**
5. **FIND OUT HOW TO PLOT THE BROKEN SYMMETRY SECTOR OF THE SPECTRUM IN FIG 1.**

14/04/21
MEETING Day!
Prof Elena Ostrovskaya replied to my email and clarified that my assumption of the orthogonality of the eigenstates is not true.
I should find out more about these kinds of basis states.

Elena Ostrovskaya elena.ostrovskaya@anu.edu.au via anu365.onmicrosoft.com 16:23 (3 hours ago)
to Ana

Dear Ana,

Thank you for your question. In my understanding, the complex eigenstates of a non-Hermitian system are, in general, not orthogonal. I do not remember why we still had to specify that they are normalized, but I can try to dig out the calculations and check.

Best Regards,
Elena

From: Ana Fabela Hinojosa <acfab1@student.monash.edu>
Date: Wednesday, 14 April 2021 at 11:18 am
To: Elena Ostrovskaya <elena.ostrovskaya@anu.edu.au>
Subject: A question about "q" in "Observation of non-Hermitian degeneracies in a chaotic exciton-polariton billiard"

Dear Prof. Ostrovskaya,
My name is Ana, I am an undergraduate student in Monash university.
Currently I am working in a research project on non-Hermitian quantum mechanics. At the suggestion of my supervisors (Dr Jesper Levinsen and Prof. Meera Parish) I read your paper "Observation of non-Hermitian degeneracies in a chaotic exciton-polariton billiard" and I found it extremely interesting! Nevertheless I was confused about a specific definition in the methods and I was hoping that you could clarify it for me.
The definition in question concerns the mode coupling term q, defined as

$$q \propto \int \varphi_n^*(\boldsymbol{r})\varphi_{n'}(\boldsymbol{r})\,\mathrm{d}^2\boldsymbol{r}$$

This definition is confusing me because I assumed that the basis vectors $|\phi n'\rangle$,$|\phi n\rangle$ are supposed to be orthogonal (and similarly for their conjugates).
Sorry for bothering you with this, and hopefully this question doesn't take too much of your time.

Sincerely,
Ana

In my meeting with J&M I showed them diagonalisation of the 2x2 Hamiltonian (H) in Gao et al.
Since I am not super savvy using Mathematica, Jesper was able to show me how to obtain a simpler looking result for the eigenvalues of H, by setting the assumption for all our parameters to be real valued.

```
In[3]:= mat = {{λ1 - I Γ1, q}, {q, λ2 - I Γ2}};
        mat // MatrixForm
```

Out[4]//MatrixForm=

$$\begin{pmatrix} -i\,\Gamma_1 + \lambda_1 & q \\ q & -i\,\Gamma_2 + \lambda_2 \end{pmatrix}$$

Diagonalizing H:

```
In[5]:= vecs = Eigenvectors[mat];
        Inverse[Transpose[vecs]].mat.Transpose[vecs] // Chop

In[7]:= FullSimplify[%, Assumptions → q > Γ1 > Γ2 > λ1 > λ2 > 0]
```

Out[7]=

$$\left\{\left\{\frac{1}{2}\left(-i\,\Gamma_1 - i\,\Gamma_2 + \lambda_1 - \sqrt{(2\,q + \Gamma_1 - \Gamma_2 + i\,\lambda_1 - i\,\lambda_2)\,(2\,q - \Gamma_1 + \Gamma_2 - i\,\lambda_1 + i\,\lambda_2)} + \lambda_2\right), 0\right\}, \left\{0, \frac{1}{2}\left(-i\,\Gamma_1 - i\,\Gamma_2 + \lambda_1 + \sqrt{(2\,q + \Gamma_1 - \Gamma_2 + i\,\lambda_1 - i\,\lambda_2)\,(2\,q - \Gamma_1 + \Gamma_2 - i\,\lambda_1 + i\,\lambda_2)} + \lambda_2\right)\right\}\right\}$$

This output is now manageable so I can continue reading the methodology that Bender outlines.
Another suggestion made by Jesper was that I could re-scale the matrix by setting q = Eiφ or simply q = 1.

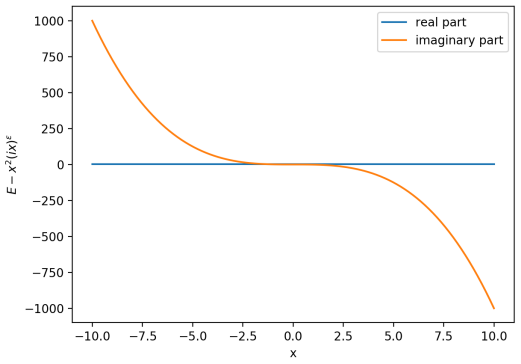
I also showed J&M my work of Bender's figure 1. Since my results are so similar and yet quite different to Bender's several points were Raised by J&M:

1. I should aim to understand the contour of integration mentioned in the text since we are dealing with integration in the complex plane and the integrand involves a square root it is likely that python is spitting out one of multiple results at random … ie. It could be the case that I am seeing selections of different Riemann sheets or something of that sort given that the square root function has a branch cut.
2. I should consider replicating the Runge Kutta results from Bender.
   But I reckon this option should not be a priority since I am unsure on how to implement RK using complex numbers.

Square root analysis

What exactly is going on in the integration of the square root of a complex number? Here I am plotting the behaviour of the RHS of the WKB approximation as stated in Bender. First I want to check what is going on with my integrand without the square root for any x and some fixed ϵ, (note:ϵ=1, E0 = 1.1563):

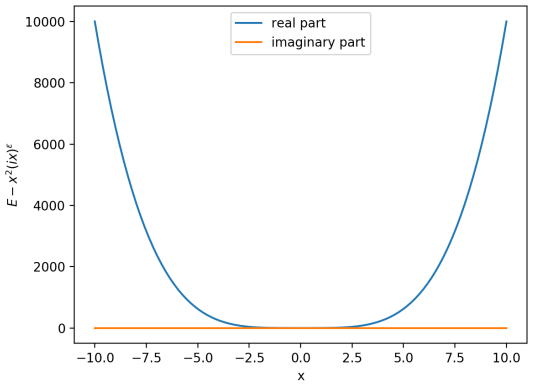
```
########################TEST#########################################
def whats_up_with_integrand(x_values, E, ϵ):
    # checking the direction of rotation of the non-Hermitian part of the potential
    # for some x values
    complex_numbers = []
    for x in x_values:
        complex_num = E - x**2 * (1j * x)**ϵ
        complex_numbers.append(complex_num)

    plt.plot(x_values, np.real(complex_numbers), label="real part")
    plt.plot(x_values, np.imag(complex_numbers), label="imaginary part")

    plt.legend()
    plt.ylabel(r'$E - x^2 (i x)^\epsilon $')
    plt.xlabel('x')
    # plt.title("")
    plt.show()

x_values = np.linspace(-10, 10, 100)
# ϵ = np.linspace(-0.5, 3, 30)
ϵ = 1
whats_up_with_integrand(x_values, E0, ϵ)
########################TEST#########################################
```



In the next plot ! am simply increasing $\epsilon$by 1 in order to see if there are any strange behaviours (ie. discontinuities).

It looks like the real and imaginary parts swap from action/inaction (one of them is zero everywhere and the other one isn't) depending on the power ϵ(even or odd behaviour, This only occurs if the power is an integer).
I observe no discontinuous behaviour so far.

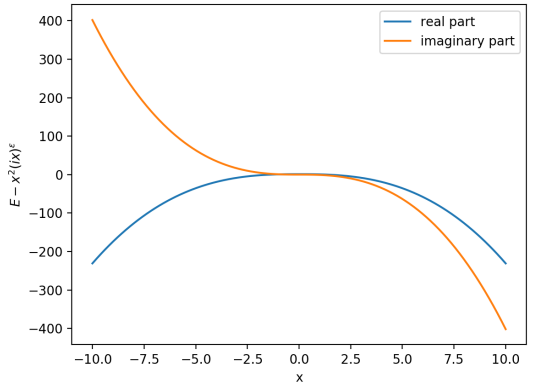
ϵ=2



As is visible here, if the power is not an integer both parts of the integrand's argument are "active" even though they do not share oddness or evenness.

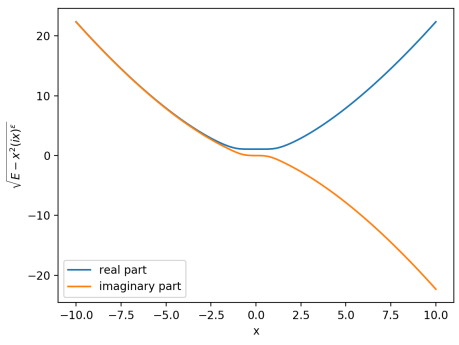
What happens if I take the square root? And observe the actual integrand in Bender?
Will I have jumpy behaviour in my plots?

ϵ=⅔

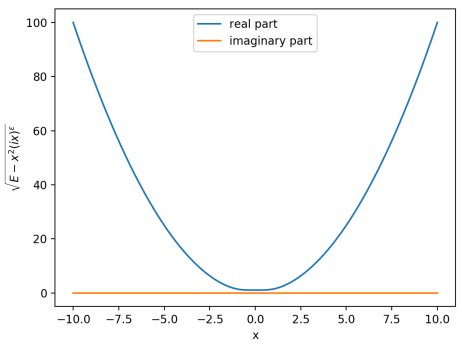


No, even after implementing the square root function, the plots **do not show discontinuities** as far as I can see.

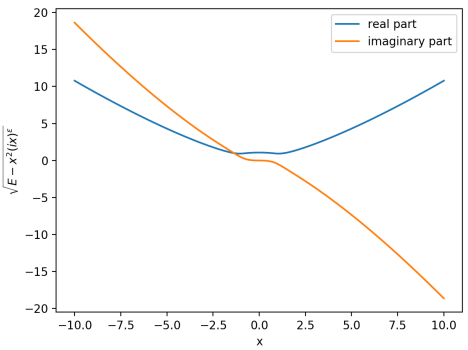
ϵ=1



ϵ=2



ϵ=⅔



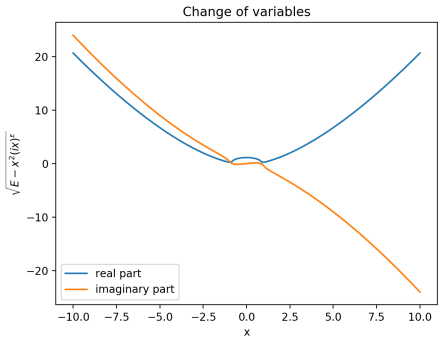
I am pretty certain that this demonstrates that:
Python would not be giving me an answer to the square root chosen at random IF I was using Bender's integrand.
16/04/21

Given that I am using a change or variables in my integral I should do the same sort of test but for **the integrand that I am actually using in my code.**

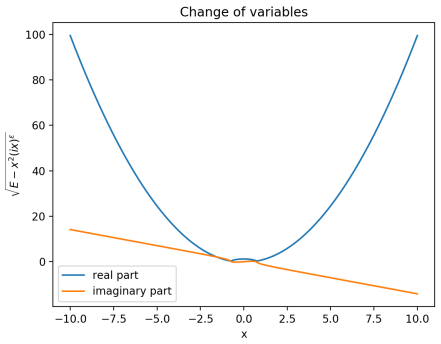
```
########################TEST#####################################
def whats_up_with_integrand3(x_values, E, ϵ):
    # checking the direction of rotation of the non-Hermitian part of the potential
    # for some x values using the change of variables from my integrand() function

    α = 3/2 - 1/(ϵ +2)
    offset = - 1j * np.sin(np.pi * (3/2 - 1/(ϵ +2)))

    complex_numbers = []
    for x_prime in x_values:
        x = x_prime - offset
        complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)
        complex_numbers.append(complex_num)

    plt.plot(x_values, np.real(complex_numbers), label="real part")
    plt.plot(x_values, np.imag(complex_numbers), label="imaginary part")

    plt.legend()
    plt.ylabel(r'$\sqrt{E - x^2 (i x)^\epsilon}$')
    plt.xlabel('x')
    plt.title("Change of variables")
    plt.show()

x_values = np.linspace(-10, 10, 10000)
ϵ = 1
whats_up_with_integrand3(x_values, E0, ϵ)
########################TEST#####################################
```

No, the plots **do not show discontinuities** as far as I can see.

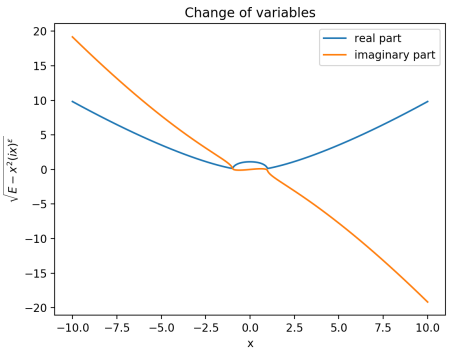
ϵ=1

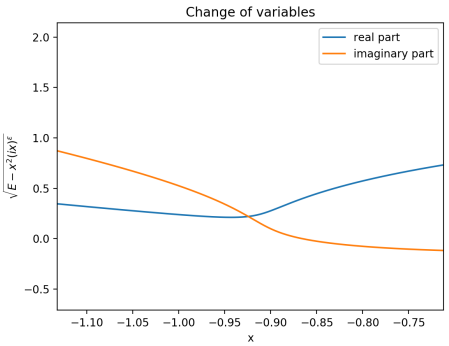


ϵ=2



ϵ=⅔





Zoomed version of the plot for the case when ϵ=1 above.
The plots are continuous and smooth everywhere in sight.

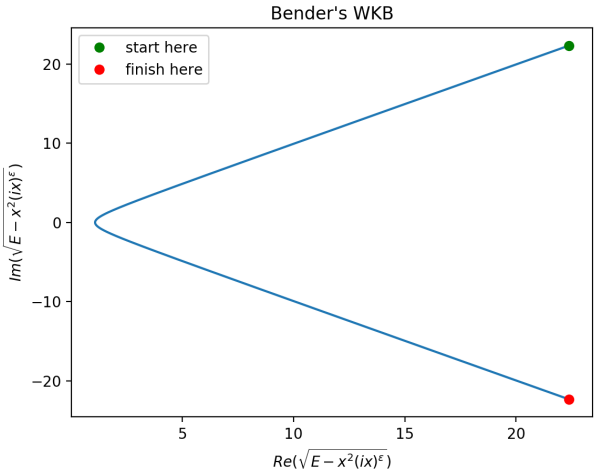
Even though my plots look slightly different from the ones yielded by Bender's expression (complex valued) I am pretty certain that this demonstrates that **Python is not giving me an answer to the square root that is chosen at random (from distinct Riemann sheets)**.

18/04/21

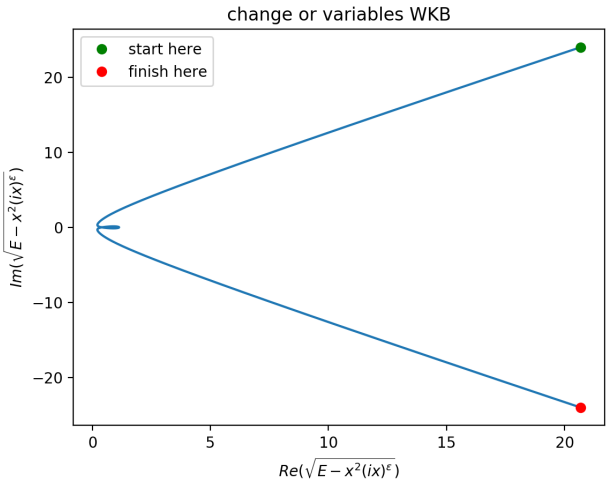
Below I show that as x increases the resulting values in the integrand form a symmetric continuous function in the positive real quadrants of the argand plane, First I do the analysis for the WKB expression in Bender, and immediately below I modify the code for my change of variables below

It can be seen that the trajectory begins in the positive imaginary, and positive real axes (first quadrant) crosses at the origin and continues to the negative imaginary, and positive real axes (fourth quadrant).

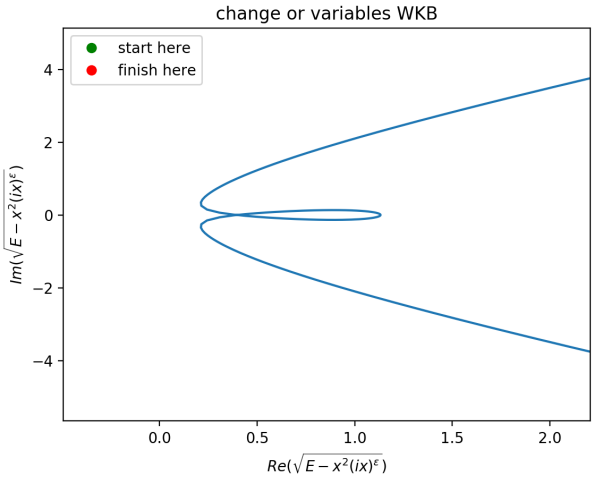
```
#########################TEST#########################################
def whats_up_with_integrand(x_values, E, ϵ):
    # checking the direction of rotation of the non-Hermitian part of the potential
    # for some x values
    reals = []
    imaginary = []
    for x in x_values:
        complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)
        reals.append(np.real(complex_num))
        imaginary.append(np.imag(complex_num))

    plt.plot(reals, imaginary, '-')
    plt.plot(reals[0], imaginary[0],'go', label='start here')
    plt.plot(reals[-1], imaginary[-1],'ro', label='finish here')
    plt.legend()
    plt.ylabel(r'$Im(\sqrt{E - x^2 (i x)^\epsilon})$')
    plt.xlabel(r'$Re(\sqrt{E - x^2 (i x)^\epsilon})$')

    plt.title("Bender's WKB")
    plt.show()

x_values = np.linspace(-10, 10, 1024)
# ϵ = np.linspace(-0.5, 3, 30)
ϵ = 1
whats_up_with_integrand(x_values, E0, ϵ)
```




```
#########################TEST#########################################
def whats_up_with_integrand(x_values, E, ϵ):
    # checking the direction of rotation of the non-Hermitian part of the potential
    # for some x values
    α = 3/2 - 1/(ϵ +2)
    offset = - 1j * np.sin(np.pi * (3/2 - 1/(ϵ +2)))

    reals = []
    imaginary = []
    for x_prime in x_values:
        x = x_prime - offset
        complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)
        reals.append(np.real(complex_num))
        imaginary.append(np.imag(complex_num))

    plt.plot(reals, imaginary, '-')
    plt.plot(reals[0], imaginary[0],'go', label='start here')
    plt.plot(reals[-1], imaginary[-1],'ro', label='finish here')
    plt.legend()
    plt.ylabel(r'$Im(\sqrt{E - x^2 (i x)^\epsilon})$')
    plt.xlabel(r'$Re(\sqrt{E - x^2 (i x)^\epsilon})$')

    plt.title("change or variables WKB")
    plt.show()

x_values = np.linspace(-10, 10, 1000)
# ϵ = np.linspace(-0.5, 3, 30)
ϵ = 1
whats_up_with_integrand(x_values, E0, ϵ)
```



The function is still continuous and smooth but it shows even more multivalue behaviour than Bender's expression.



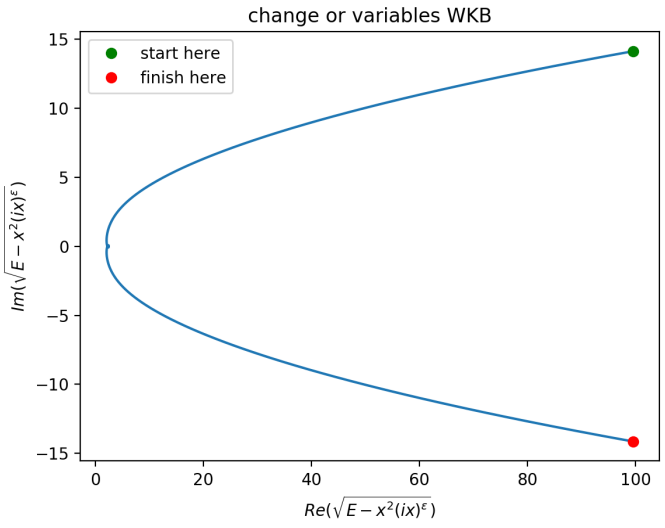
This path is the integration path that Bender describes in the methodology describing the BCs that define the Schrödinger equation.

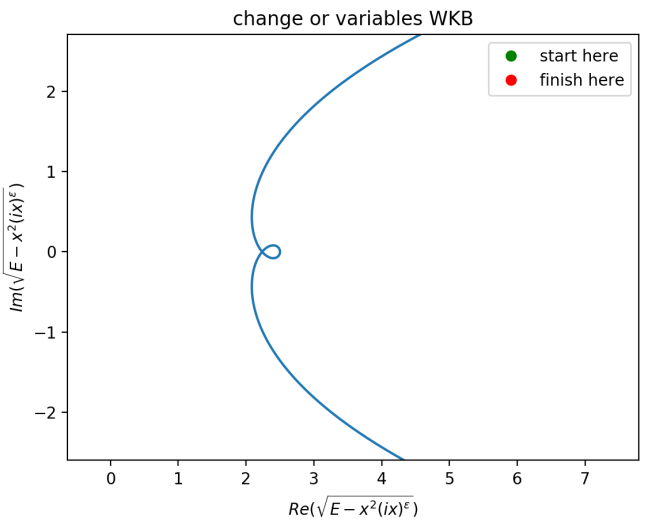
NEXT:

PLOT a different E for a corresponding Є

For the case

ϵ = 2, n =1, E = 6.003386082





Important questions:

- Is the path I see the one Bender describes as the integration contour lying inside the Stokes wedges in the complex plane?
- Why is my change of variables showing this strange loop behaviour near the origin?

Week 7 (19/04/21 – 25/04/21)

Aims:

1. Reproduce Fig 1 in Bender
2. Compare Gao et al.'s (2x2 Hamiltonian) to Bender page 986 (is the comparison useful?)
3. Simulate a RF-CAP

Tasks:

1. ~~Continue TESTS_fig 1 in Bender (unbroken symmetry sector)~~
2. **FIND OUT HOW TO PLOT THE BROKEN SYMMETRY SECTOR OF THE SPECTRUM IN FIG 1.**
3. Continue Gao's Hamiltonian analysis (using Benders methodology)
4. **Understand how is it that complex eigenstates of NH systems are not orthogonal**
    a. Maybe something to do with Bi-orthogonality?

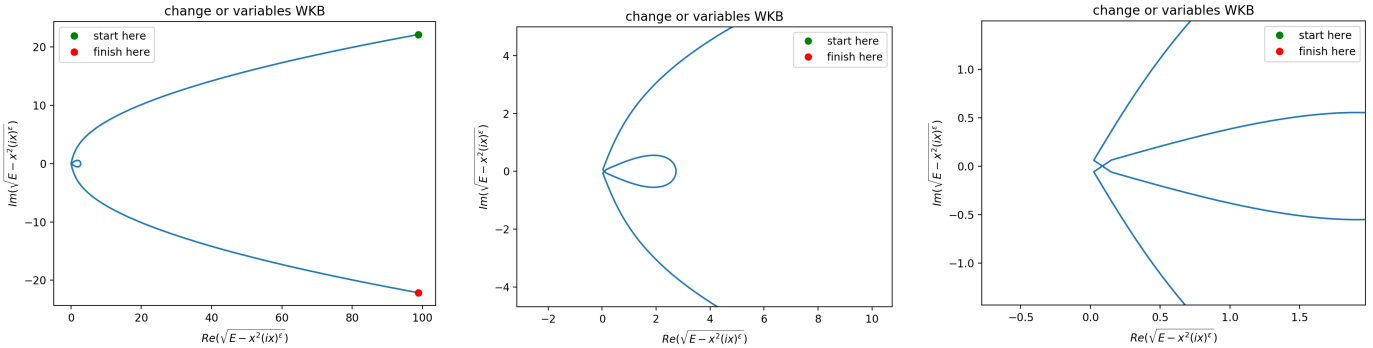
20/04/21

Square root analysis (continued)

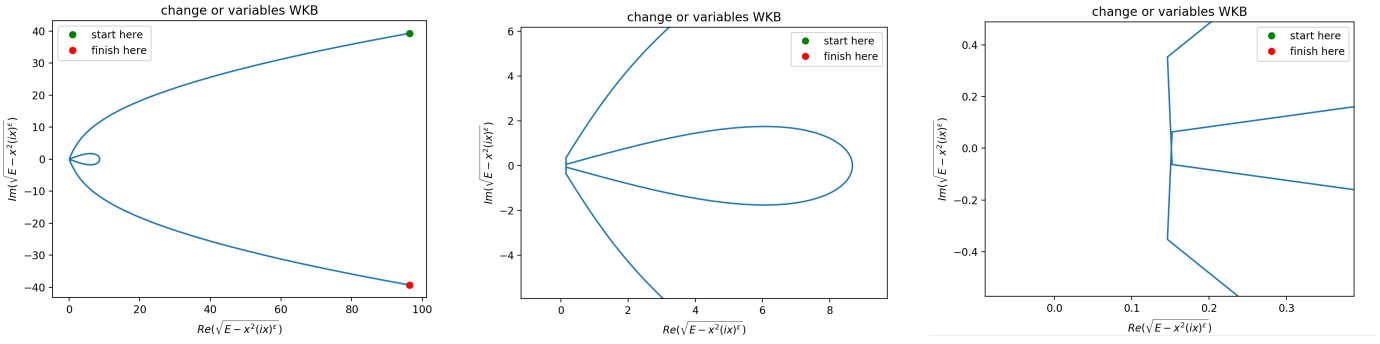
I've noticed that my change of variables offset was still missing a factor of: $E^{(1/(\epsilon+2))}$ (I had noticed this error before but failed to make this correction)

I have made this correction and this is the resulting plot (zoomed) for the cases (as reported in Table 1. In Bender)

ϵ = 2, n =1, E = 6.003386082



ε = 2, n = 8, E = 60.185767651



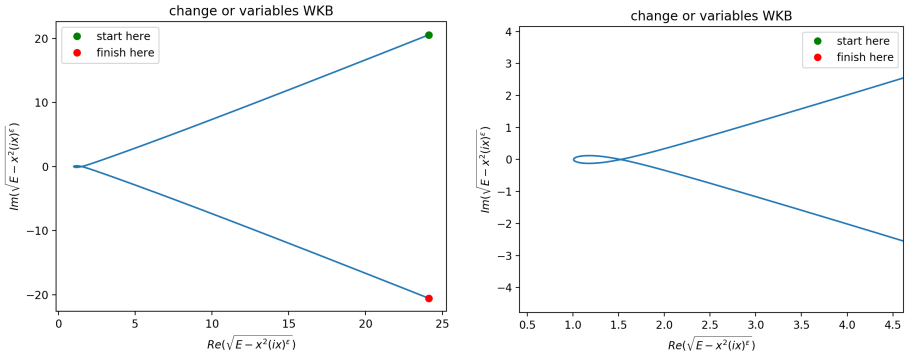
What happens if I change the sign of my offset in the change of variables?
Currently I am using

```
α = 3/2 - 1/(ϵ +2)
offset = - E**(1/(ϵ+2)) * 1j * np.sin(np.pi * α)
```

```
for x_prime in x_values:
    x = x_prime - offset
    complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)
```

So I'll change the offset to a positive number

ϵ = 1, n = 0, E0 = 1.1563

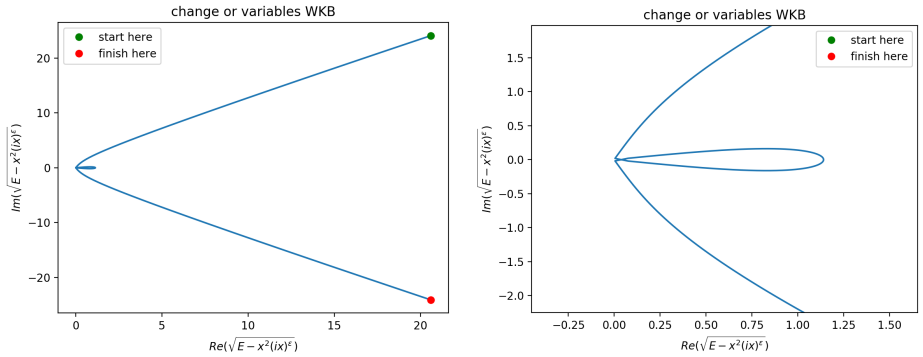


The loop still appears… just by the looks of the plot I suspect that the negative sign offset was closer to the desired result.

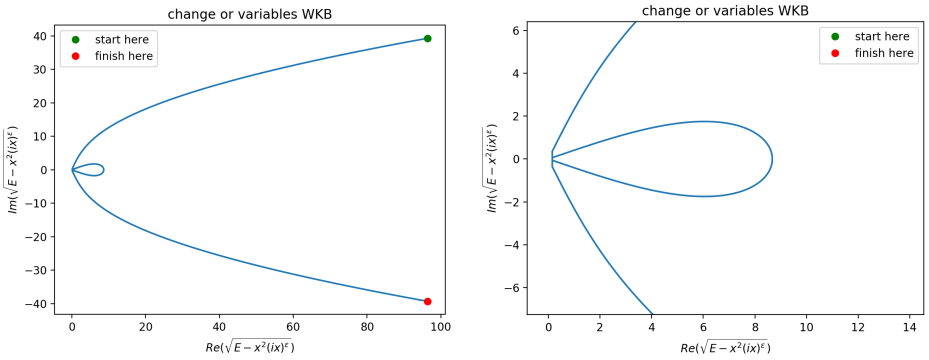
I changed the offset once again but this time I used the argument from the x_+ limit ie:

```
β = 1/2 - 1/(ϵ +2)
offset = + E**(1/(ϵ+2)) * 1j * np.sin(np.pi * β)
```

As expected, given the symmetry of the setup, the result from using this argument is very similar to the previous one
ϵ = 1, n = 0, E0 = 1.1563



ε = 2, n = 8, E = 60.185767651



Why is the contour of integration looping near the origin when I do a change of variables? Clearly something subtle is wrong because I cannot find the issue . Presently, I cannot think of other possible tests to apply to my methodology.
I must continue onwards and try to understand what is going on in the broken symmetry part of the spectrum.

> When I continued reading Bender I found this…
> There are several highly accurate numerical techniques for computing the energy spectrum that is displayed in figure 1. The simplest and most direct method is to integrate the Schrödinger differential equation (24) using a Runge–Kutta approach. To do so, we convert this complex differential equation to a system of coupled, real, second-order equations (Bender 962)...
> The convergence is most rapid when we integrate along paths located at the centers of the Stokes wedges and follow these paths out to ∞. We then patch the two solutions in each Stokes wedge together at the origin.(Bender 963).

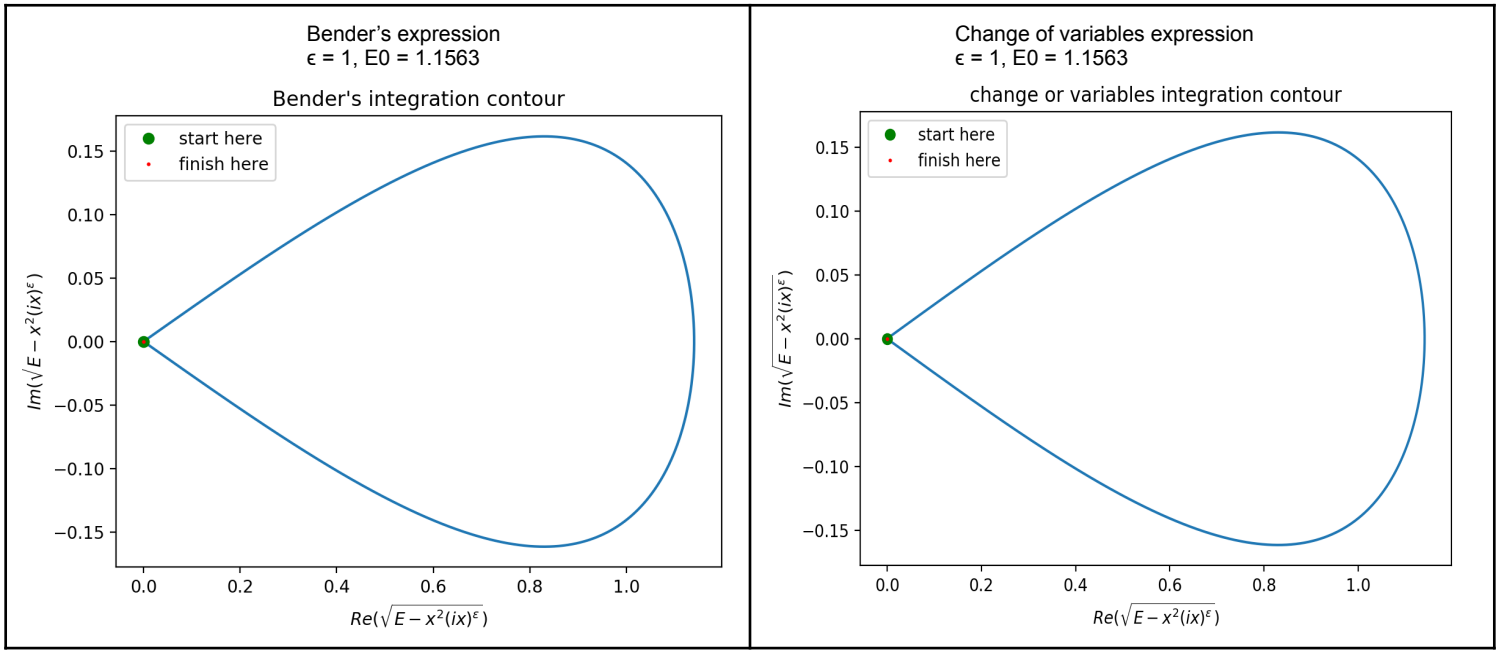
Maybe I should follow Meera's advice and try using Runge-Kutta instead of a change of variables WKB
I could solve the real and imaginary parts of the equation separately and use the shooting method

Turns out that I was feeding the wrong linspace object to my function and to Bender's expression when I was checking the integration contour… in addition I was missing a bunch of i's in my change of variables(in the integrand and the limits). I assume that this happened because I was writing the analytic expressions that I worked out in paper rather than using the functions provided in python to add and subtract the imaginary offset of my change of variables.
I was also plotting the expression from Bender's paper wrong because I had shifted my x-values from the complex plane to the real line when that expression didn't require the transformation of the x-values at all.
I now know that Python doesn't spit out random results from taking the square root! But instead has a sensical method that returns a continuous set of roots.

Eg.
tp_minus = E(1/(ϵ+2)) * np.exp(np.pi * (3/2 - (1/(ϵ+2)))) vs tp_minus = E**(1/(ϵ+2)) * np.exp(1j * np.pi * (3/2 - (1/(ϵ+2))))**

I have the corrected code and below the code snippet I show the plot that results from Bender's expression and from my change of variables

```
######################## TEST 1 ####################################
def whats_up_with_integrand(x_values, E, ϵ):
    # checking the integration path of the integrand in the x-complex plane
    reals = []
    imaginary = []
    for x in x_values:
        complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)

        reals.append(np.real(complex_num))
        imaginary.append(np.imag(complex_num))

    plt.plot(reals, imaginary, '-')
    plt.plot(reals[0], imaginary[0],'go', label='start here')
    plt.plot(reals[-1], imaginary[-1],'ro', markersize='1.2', label='finish here')
    plt.legend()
    plt.ylabel(r'$Im(\sqrt{E - x^2 (i x)^\epsilon})$')
    plt.xlabel(r'$Re(\sqrt{E - x^2 (i x)^\epsilon})$')

    # plt.title("Bender's integration contour")
    plt.title("change or variables integration contour")
    plt.show()


# Bender's integral
x_values = np.linspace(tp_minus, tp_plus, 10000)

# change of variables integral
x_values = np.linspace(tp_minus_prime, tp_plus_prime, 10000) + 1j * np.imag(tp_minus)

whats_up_with_integrand(x_values, E0, ϵ)
######################## TEST 1 ####################################
```



Bender's expression
ϵ = 1, E0 = 1.1563

Change of variables expression
ϵ = 1, E0 = 1.1563

WOOHOO! Victory! The plots are identical and I don't see any sort of branching behaviour in the contour.

I should check if my bug fixes help with my integration issues.

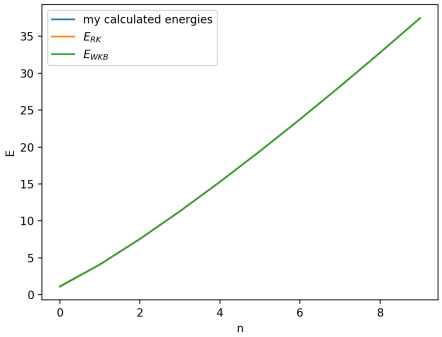
I wrote the following globals, and corrected the integrand function (which was missing a factor of i on the offset) for the following test

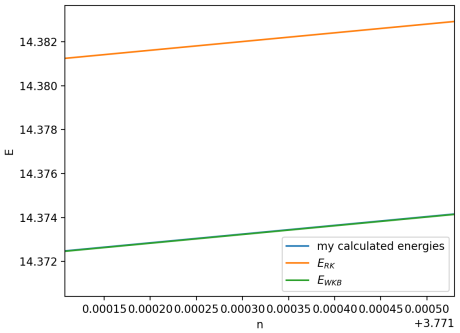
```
#################################function calls#####################################################

#IC based on RK results give (ϵ, n) = (1, 0)
ϵ = 1
E0 = 1.1563
E = E0

tp_minus = E**(1/(ϵ+2)) * np.exp(1j * np.pi * (3/2 - (1/(ϵ+2))))
tp_plus = E**(1/(ϵ+2)) * np.exp(-1j * np.pi * (1/2 - (1/(ϵ+2))))
tp_minus_prime = E**(1/(ϵ+2)) * np.exp(1j * np.pi * (3/2 - (1/(ϵ+2)))) - 1j * np.imag(tp_minus)
tp_plus_prime = E**(1/(ϵ+2)) * np.exp(-1j * np.pi * (1/2 - (1/(ϵ+2)))) - 1j * np.imag(tp_minus)
```

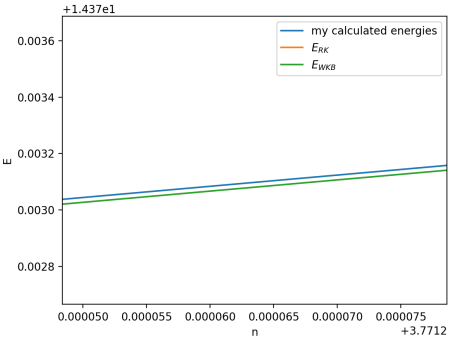
```
def integrand(x_prime, tp_minus, E, ϵ):
    # Change of variables integrand
    x = x_prime + 1j * np.imag(tp_minus)
    return np.sqrt(E - x**2 * (1j * x)**ϵ)
```

```
######################## TEST 2 ####################################
# iterative approach 1 for ϵ = 1
Energies_1 = []
for n in range(10):
  E = complex_fsolve(error, E0, args=(1, n))
  Energies_1.append(E)

# comparison to WKB and RK reported in Bender
n = range(10)
E_RK= [1.1563, 4.1093, 7.5623, 11.3144, 15.2916, 19.4515, 23.7667, 28.2175, 32.7891, 37.4698]
E_WKB = [1.0943, 4.0895, 7.5489, 11.3043, 15.2832, 19.4444, 23.7603, 28.2120, 32.7841, 37.4653]
plt.plot(n, Energies_1 , label="Energies_2")
plt.plot(n, E_RK , label=r"$E_{RK}$")
plt.plot(n, E_WKB , label=r"$E_{WKB}$")
plt.legend()
plt.xlabel("n")
plt.ylabel("E")
plt.show()
```

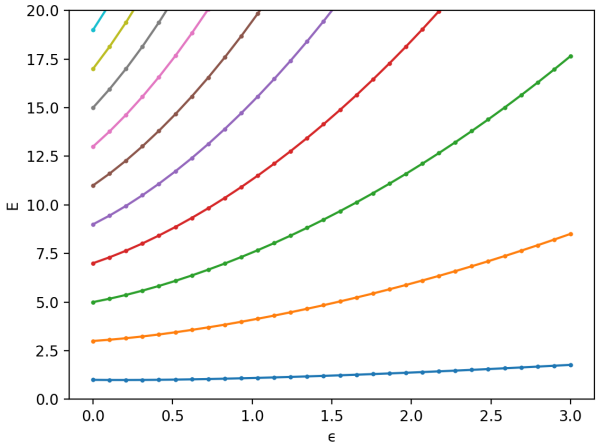
Output:







I have near perfect agreement with the values reported in Bender for ($\epsilon = 1$).
This means that my WKB method works! And also that the unbroken symmetry part of the spectrum is now correct!



21/04/21
I suspect that the way to go with trying to plot the broken symmetry part of the spectrum is going to be using Runge-Kutta. I haven't found any other mention of the methodology used by Bender for this sector of the spectrum.

23/04/21

Plotting Figure 1 with Runge-Kutta

To integrate the complex Schrödinger equation

$$\text{let } \hbar = 1,\ m = \tfrac{1}{2}$$

$$\hat{p}^2\psi + x^2(ix)^\epsilon\psi = E\psi$$

$$\left(-i\tfrac{d}{dx}\right)^2\psi + x^2(ix)^\epsilon\psi = -\tfrac{d^2}{dx^2}\psi + x^2(ix)^\epsilon\psi = E\psi$$

using a Runge–Kutta approach. I must convert the complex equation to a system of coupled, real, second-order equations. I don't know when I exactly this needs to occur because to solve the equation in the first place I need ICs and BCs and I am unsure on how to obtain these.
First, I need to find my ICs to feed into the RK method, to find these I need to use an asymptotic solution, But I don't know that this will work.

Asymptotic analysis

If: $x \ll -1$ TRY: $\psi = Ae^{ikx} + Be^{-ikx}$

The equation terms for small x are

$$-\tfrac{d^2}{dx^2}\psi = (ik)^2Ae^{ikx} + (-ik)^2Be^{-ikx} = -k^2Ae^{ikx} - k^2Be^{-ikx} \sim -k^2Be^{-ikx}$$

$$x^2(ix)^\epsilon\psi = x^2(ix)^\epsilon(Ae^{ikx} + Be^{-ikx}) \sim x^2(ix)^\epsilon Be^{-ikx}$$

$$E\psi = E(Ae^{ikx} + Be^{-ikx}) \sim EBe^{-ikx}$$

$$\therefore -\tfrac{d^2}{dx^2}\psi + x^2(ix)^\epsilon\psi = E\psi$$

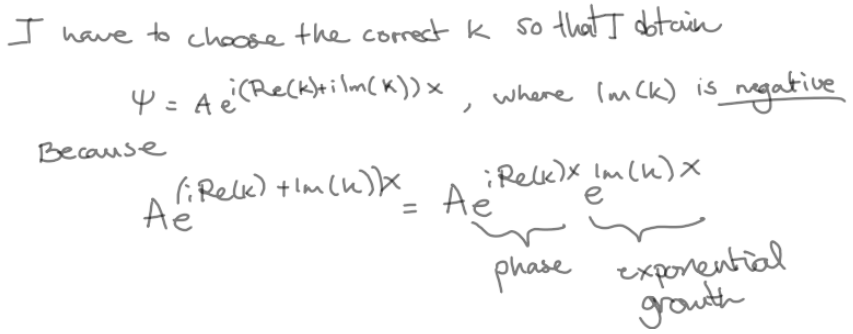
$$\sim -k^2Be^{-ikx} + x^2(ix)^\epsilon Be^{-ikx} = EBe^{-ikx}$$

Solving this equation for k

$$\therefore -k^2 + x^2(ix)^\epsilon = E$$

$$\therefore k = \sqrt{x^2(ix)^\epsilon - E}$$

I would have to choose the correct k so that I obtain a IC wavefunction

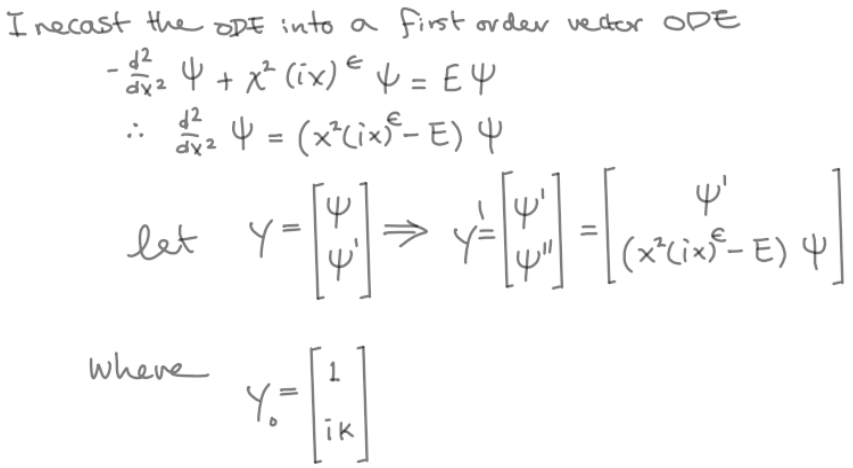



```
def find_k(x, ϵ, E):
    return np.sqrt(x**2 * (1j * x)**ϵ - E)

# Asymtotic solution for k
x_asymptotic = -30
# values from Bender table 1.
ϵ1 = 1
E1 = 1.1563

print("\n",find_k(x_asymptotic, ϵ1, E1), "\n")
```

```
(116.18701245139194-116.1919883743277j)
```

This is a lucky first result… (find_k gave me the answer I wanted randomly yay)

Solving the Schrödinger equation requires that I recasut the 2nd order ODE into a first order vector system. The thing is that the system is still complex here so I am not sure when I need to split it into 2 real coupled equations.




```
# Schrödinger equation
def Schrodinger_eqn(x, Ψ):
    psi, psi_prime =  Ψ
    psi_primeprime = (x**2 * (1j * x)**ϵ - E) * psi
    Ψ_prime = np.array([psi_prime, psi_primeprime])
    return Ψ_prime
```

```
# k values
k = find_k(x_asymptotic, ϵ, E)
# IC
Ψ0 = [1, 1j * k]
```

24/04/21

I am having a hard time conceptualising the use of Runge Kutta and eventually solving the Schrödinger equation. I am finding it tricky because my equation is complex valued, so I am unsure of the order and process by which I should separate real and imaginary parts to solve the equation.

Week 8 (26/04/21 – 02/05/21)

Aims:

1. Reproduce Fig 1 in Bender
2. **Understand how is it that complex eigenstates of NH systems are not orthogonal**
    a. Maybe something to do with Bi-orthogonality?
3. Simulate a RF-CAP

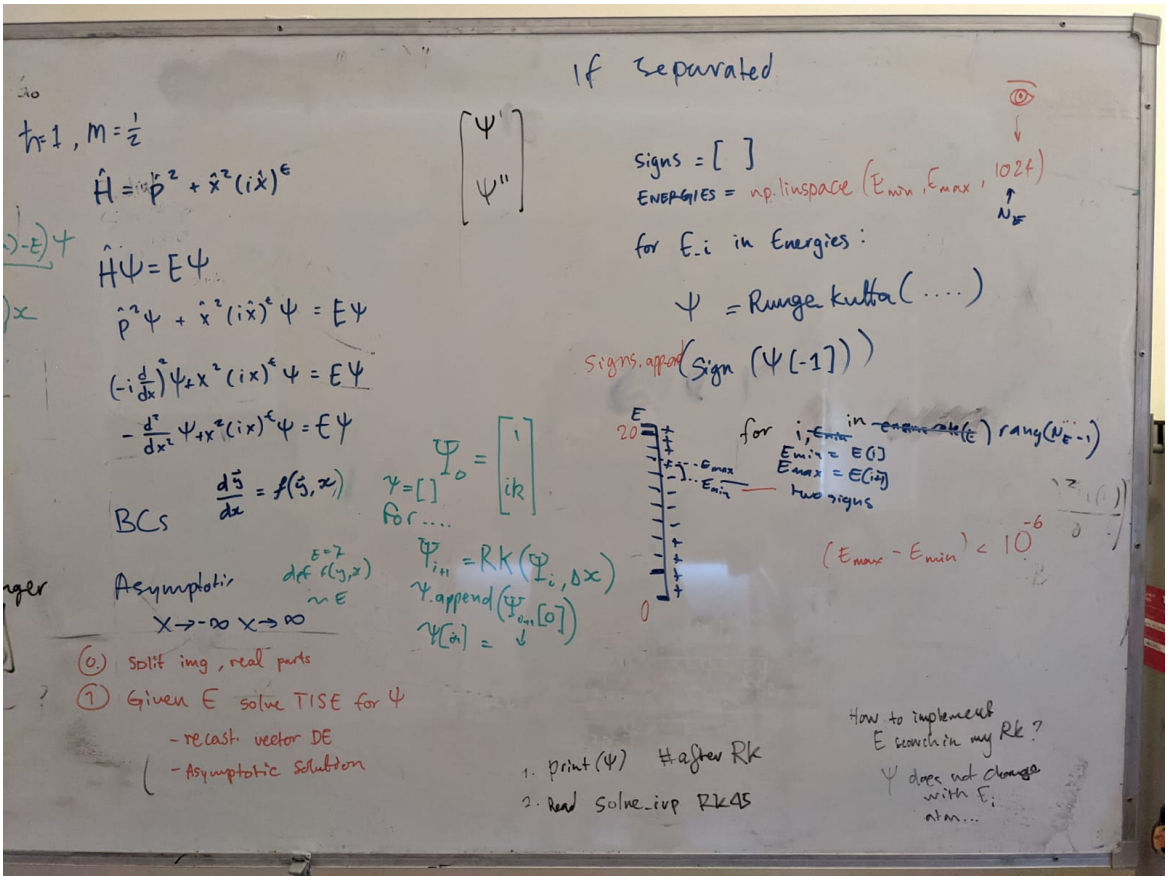
Tasks:

1. ~~Continue TESTS fig 1 in Bender (unbroken symmetry sector)~~
2. ~~**FIND OUT HOW TO PLOT THE BROKEN SYMMETRY SECTOR OF THE SPECTRUM IN FIG 1.**~~
   HOW TO SEPARATE MY EQUATION INTO TWO COUPLED EQUATIONS?
3. Continue Gao's Hamiltonian analysis (using Benders methodology)
4. ~~START progress report Due: Friday 7th May (week 9)~~

26/04/21

Plotting Figure 1 with Runge-Kutta (continued)

So, I've been thinking about it quite a bit and I am currently building a shooting method that uses Runge Kutta to find the solution to the Schrödinger equation. The issue with the complex Hamiltonian remains though, I am unsure if this method will work at all.
This is a photo of my whiteboard:



Another aspect of this problem that I am unsure about is which BCs I will use to fully solve the Schrödinger equation.
Currently this is my Schrödinger equation and Runge_Kutta, after I implemented a changing E value as input (for the shooting method):

```python
# Schrödinger equation
def Schrodinger_eqn(x, E, ϵ, Ψ):
    u, v =  Ψ
    w = (x**2 * (1j * x)**ϵ - E) * u
    Ψ_prime = np.array([v, w])
    return Ψ_prime

def Runge_Kutta(x, delta_x, E, ϵ, Ψ):
    k1 = Schrodinger_eqn(x, E, ϵ, Ψ)
    k2 = Schrodinger_eqn(x + delta_x / 2, E, ϵ, Ψ + k1 * delta_x / 2)
    k3 = Schrodinger_eqn(x + delta_x / 2, E, ϵ, Ψ + k2 * delta_x / 2)
    k4 = Schrodinger_eqn(x + delta_x, E, ϵ, Ψ + k3 * delta_x)
    return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
```

27/04/21

Shooting method

The shooting method solves the TISE using Runge_Kutta and tries different values of E until we find one that causes the wavefunction to not blow up at large x. Hence finding a viable solution to the TISE (Bound state).

The ODE setup:
Since our equation is second order, we recast the ODE into a first order vector equation.

```
# Schrödinger equation
def Schrodinger_eqn(x, E, ϵ, Ψ):
    u, v = Ψ
    w = (x**2 * (1j * x)**ϵ - E) * u
    Ψ_prime = np.array([v, w])
    return Ψ_prime
```

The ODE solver:
The cool thing about the shooting method is that once we have integrated over all x for a specific E_i value in the range we tell the computer to only "care" about the behaviour of the solution at our upper integration limit for x: **x_max**.
Since the bound states remain small (zero) at x_max, then by the **intermediate value theorem (IVT)** the binding energies will be found between the energies of "solutions" that "blow-up" at x_max (blow-up positively and negatively).
The algorithm requires that we keep track of:
-the sign of each solution: psi(x_max)
-the energy of psi(x_max)
We can store these in 2 separate lists of the same size. (A signs list and an Energies list).

We are trying to find the eigenenergy of a solution to the ODE given a (currently complex) potential
The **steps** are as follows:

1. Using the whole energy range of the potential (this is a first guess of the range where a bound state is to be found).
    a. The bounds I used for this step are E_lower = 0 and E_upper = 20 (both boundaries are set to be dimensionless from the TISE)
    b. I discretise the range E from E_lower to E_upper in n evenly spaced steps.

2. Using this first E range I solve the ODE (iteratively: for each E value in the range) by integrating for all x.

3. After the first run at solving the ODE I tell the computer to read the list of signs and notice the index where there is a change in sign (from positive to negative or vice versa).

4. I make the computer extract the corresponding (indexed) pairs of values from the Energies list and make them into a couple of upper and lower bounds to be checked:
   [E_lower, E_uppper], since we know by the **IVT** that there is a bound statw between each of these pairs of energy values.

5. Since I have found all the subsets of energy bounds for the bound states we have to iteratively scan each subset.
    a. What we do here is unpack each of these lists **[E_lower, E_upper]**
    b. Define a midpoint between them: **E_mid = E_mid = (E_lower + E_upper) / 2**
    c. use **E_mid** to solve the ODE (once again for all x)

6. Just like before, we will have to keep track of the sign of the solutions at x_max.
7. This time we will compare these signs to the sign of the solution corresponding to E_lower and E_upper.
    a. IF the signs of these new solutions matches either of the signs corresponding to the solutions we had recorded
    b. THEN we set E_mid to either a new E_lower OR E_upper
       This allows me to resume the search in a smaller range of energies.

In my code:
w[-1] is the solution to the ODE at x_max and sign_lower is the sign of the solution with energy E_lower (from our previous solver step).

THE CONDITIONAL STATEMENT for redefining our bounds and iteratively do the search is:

```
if np.sign(w[-1]) == sign_lower:
    E_lower = E_mid
else:
    E_upper = E_mid
```

In order to end the loop we define an arbitrary condition for the distance between E_upper and E_lower(ie. abs(E_upper - E_lower) > 1e-12 * meV)
Once this condition is not satisfied the Iterative search will end and we will have obtained a pretty accurate value for the **binding energy.**

My problems with the complex valued function occur from step 3.
How do I decide which is the sign?

Let's try something

```
# DOES THIS EVEN MAKE SENSE?
# storing the energy SIGN of the real and imaginary parts of the solutions at xmax
real_signs.append(np.sign(np.real(psi[-1])))
imag_signs.append(np.sign(np.imag(psi[-1])))
solutions.append(psi)
```

```
real_signs = [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan]


imag_signs = [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
an, nan, nan, nan, nan, nan, nan, nan]
```

Clearly the concept of the sign of a complex number is not actually meaningful.
I think I need to re imagine the shooting method…

28/04/21
MEETING DAY!

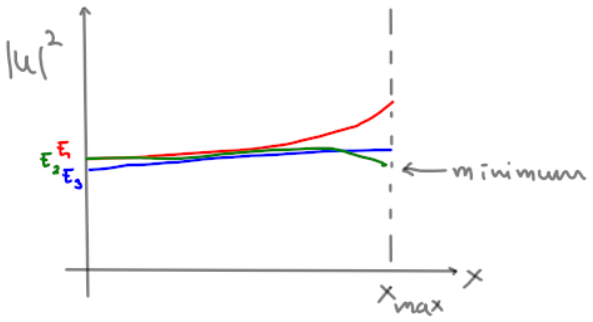
Modifying the shooting method

I talked to Chris about my issues (it always impresses me how many ideas he comes up with :)) and he suggested a cool idea
If I understood correctly… This is the strategy:

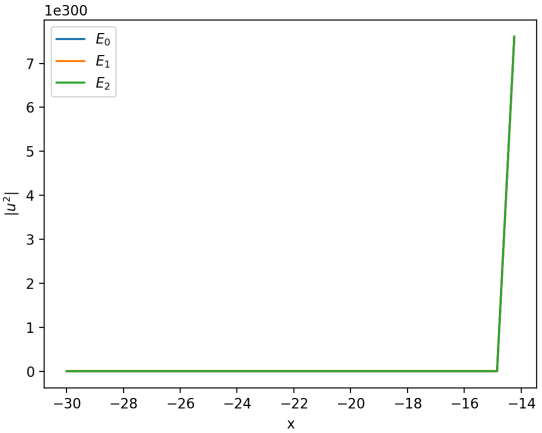
Starting with the whole energy range, I could use the mod squared (always positive) of 3 adjacent states (adjacent energy values).

1. I would still use Runge_Kutta to "evolve" the state in space as I am already doing.
2. I then store the value of the mod squared state at **x_max** in a list.
3. Then I would compare these values between any 3 adjacent states and aim to find a state such that:
   IF the the mod squared of the state between the 2 other states is less than both the mod squared values of the 2 other states
   THEN I would have found a bound state!

I should probably find a good optimization algorithm to find minima because i am finding minima



This is a plot of 3 of the mod square states I've calculated so far… they all blow up!



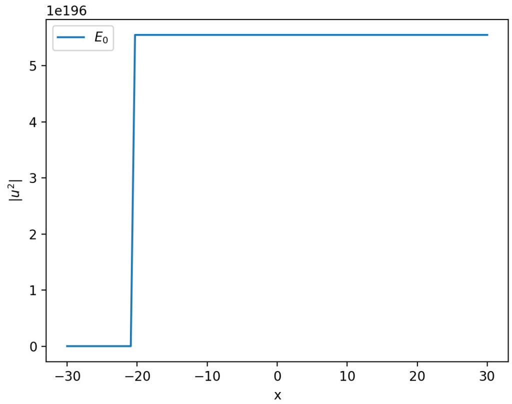
When I try to store the value of each of the mod square states at x_max I get nan values.
because the numbers are too large to be dealt with by the computer.

```
[ana:code]$ python3 other.py
other.py:70: RuntimeWarning: overflow encountered in cdouble_scalars
  mod_squared_psi.append(abs(Ψ[0]**2))
other.py:70: RuntimeWarning: invalid value encountered in cdouble_scalars
  mod_squared_psi.append(abs(Ψ[0]**2))
other.py:28: RuntimeWarning: overflow encountered in cdouble_scalars
  w = (x**2 * (1j * x)**ϵ - E) * u
other.py:37: RuntimeWarning: invalid value encountered in multiply
  return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
other.py:34: RuntimeWarning: invalid value encountered in multiply
  k2 = Schrodinger_eqn(x + delta_x / 2, E, ϵ, Ψ + k1 * delta_x / 2)
other.py:35: RuntimeWarning: invalid value encountered in multiply
  k3 = Schrodinger_eqn(x + delta_x / 2, E, ϵ, Ψ + k2 * delta_x / 2)
other.py:36: RuntimeWarning: invalid value encountered in multiply
  k4 = Schrodinger_eqn(x + delta_x, E, ϵ, Ψ + k3 * delta_x)
other.py:35: RuntimeWarning: invalid value encountered in true_divide
  k3 = Schrodinger_eqn(x + delta_x / 2, E, ϵ, Ψ + k2 * delta_x / 2)
other.py:37: RuntimeWarning: overflow encountered in multiply
  return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
other.py:37: RuntimeWarning: overflow encountered in add
  return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
other.py:28: RuntimeWarning: invalid value encountered in cdouble_scalars
  w = (x**2 * (1j * x)**ϵ - E) * u

len(mod_squared_solutions) = 100


len(mod_squared_psi) = 100


len(mod_squared_values) = 100


mod_squared_values = [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, n
an, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]
```

I need to find a way to “clip” the states after they reach a certain arbitrarily large value, but at the same time I want to be able to preserve the ordering of the states

I am trying this:

```
# Schrödinger equation
def Schrodinger_eqn(x, E, ϵ, Ψ):
    u, v =  Ψ
    # if derivative is blowing up return linear version of derivative
    if abs(v) > 1e100:
        # print("clipping blow up!")
        v = np.log(v)
        w = 0
        Ψ_prime = np.array([v, w])
        return Ψ_prime
    else:
        w = (x**2 * (1j * x)**ϵ - E) * u
        Ψ_prime = np.array([v, w])
        return Ψ_prime
```

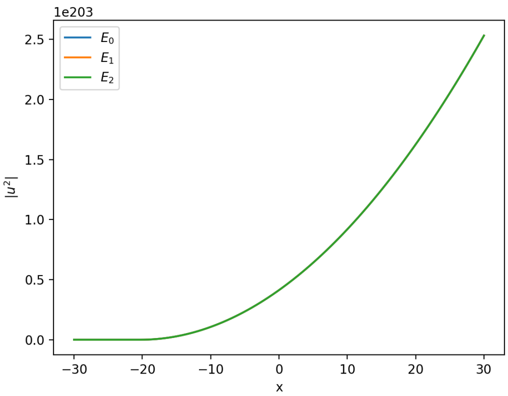


Which decreases the blow up but doesn’t work in the way I expected. I guess the log function could also give me trouble later.
Chris gave me this code snippet:

```
def abs_clip(x, level):
    """Clip the absolute value of x to
    not be greater than the given level"""
    if abs(x) > level:
        return level * x / abs(x)
    else:
        return x
```

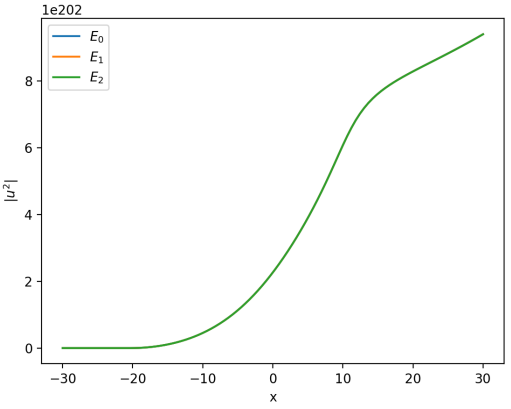
I think that it makes sense to simply scale down the states rather than change their shape and possibly create discontinuities.

```
# Schrödinger equation
def Schrodinger_eqn(x, E, ϵ, Ψ):
    u, v =  Ψ
    # if derivative is blowing up return scaled version of derivative
    if abs(v) > 1e100:
        # print("clipping blow up!")
        v = abs_clip(v, 1e100)
        w = 0
        Ψ_prime = np.array([v, w])
        return Ψ_prime
    else:
        w = (x**2 * (1j * x)**ϵ - E) * u
        Ψ_prime = np.array([v, w])
        return Ψ_prime
```



When I change a little of the redundancy conditions in the Schrödinger equation, I obtain this result:

```python
# Schrödinger equation
def Schrodinger_eqn(x, E, ϵ, Ψ):
    u, v =  Ψ
    v = abs_clip(v, 1e100)
    w = (x**2 * (1j * x)**ϵ - E) * u
    Ψ_prime = np.array([v, w])
    return Ψ_prime
```

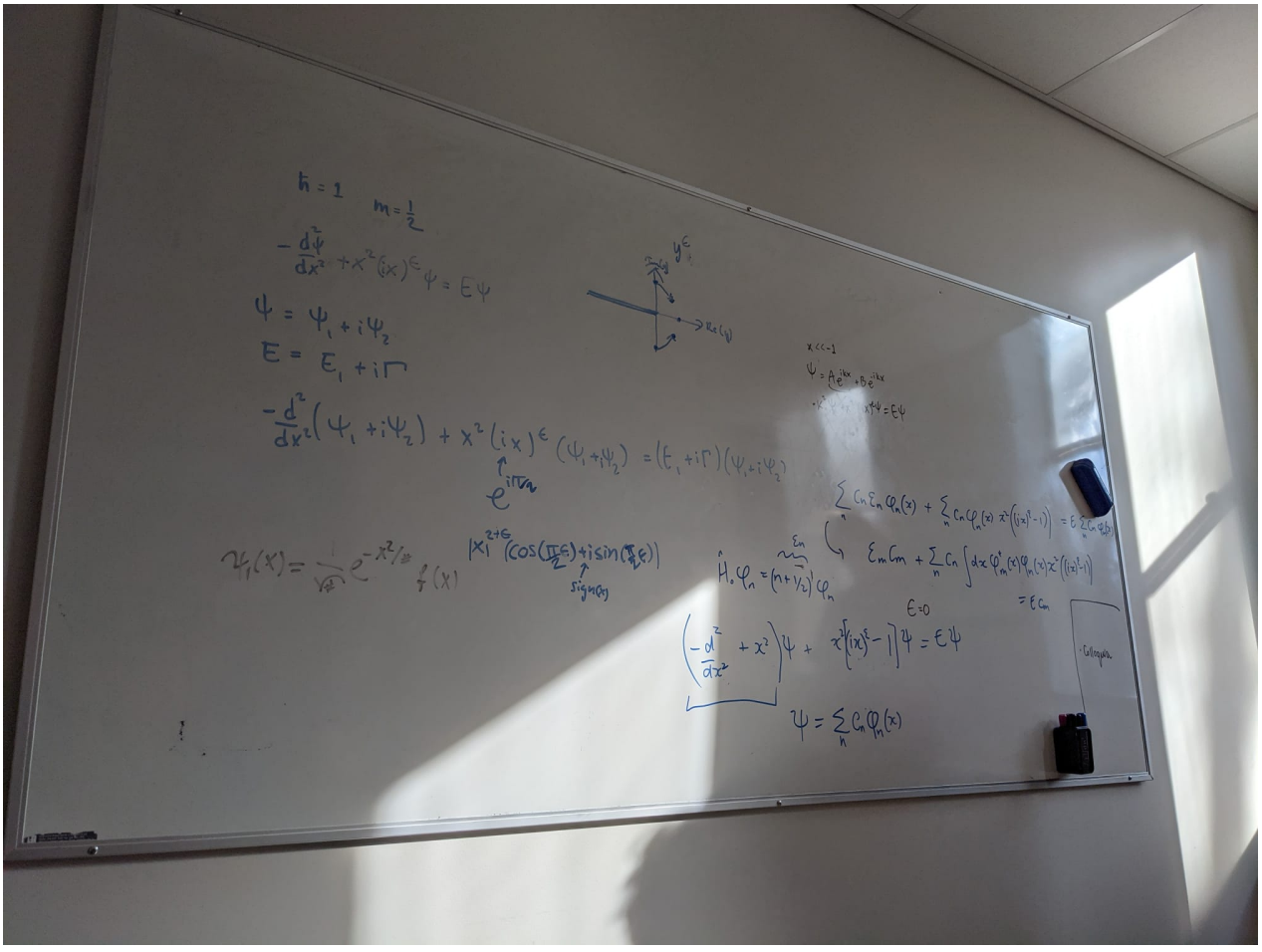


I am confused about implementing the clipping of the states. I can see that the functions blow up before even getting to x = 0
This is tricky... because if I scale the whole wavefunction down that will mess up my comparison of wavefunctions at x_max, since they will have been scaled possibly a different number of times each!

UGH! This method is probably not going to work out..

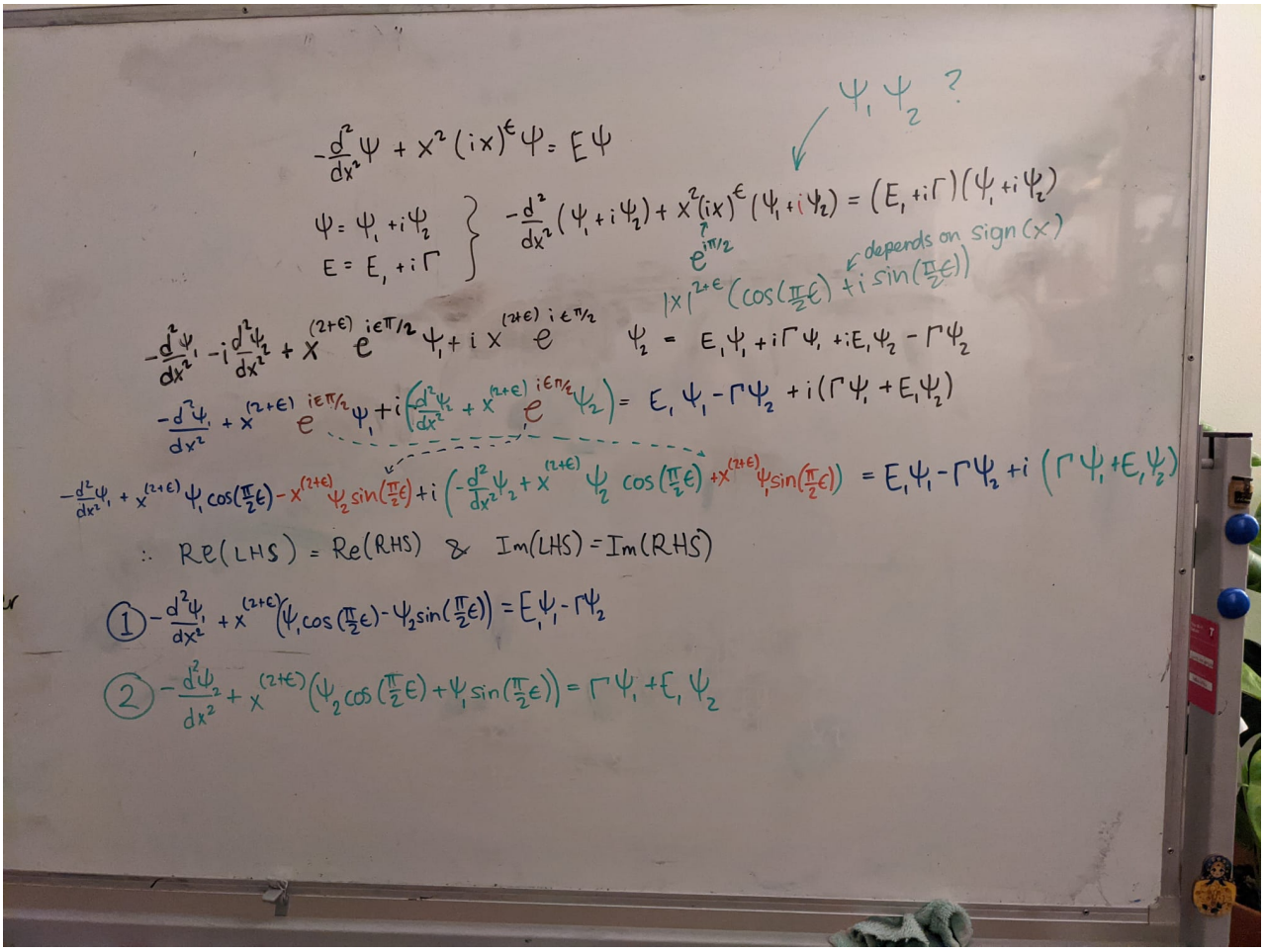
post-meeting notes:

Chatting to J&M today was great!
They suggested a method to split my Schrödinger equation into two coupled equations.



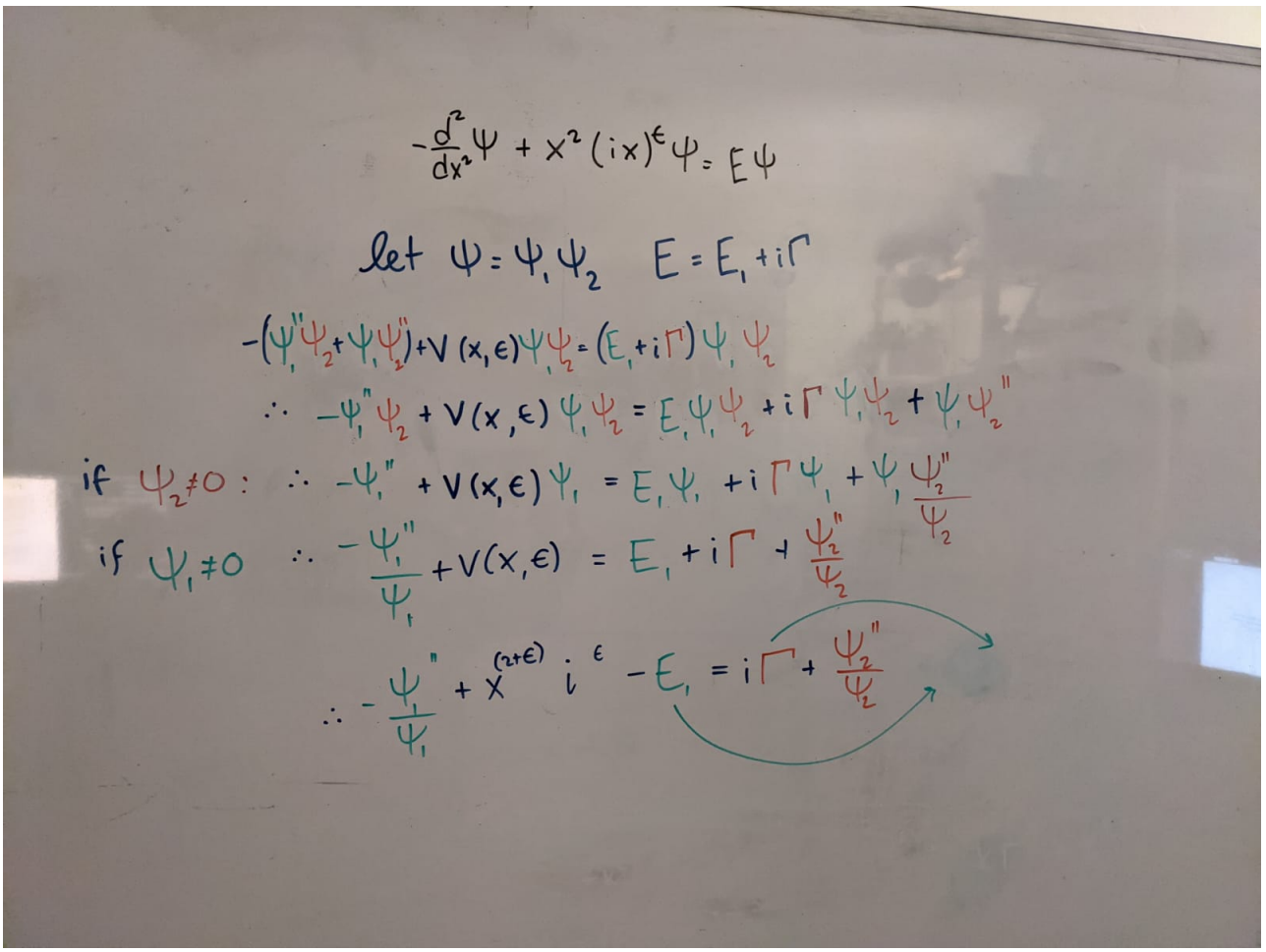
They said that I should write i as $e^{i\Pi/2}$ and to pay special attention to the value of x and its sign.
Another thing they said was to reconsider my asymptotic solution. They suggested to use the Harmonic oscillator case as the ICS

I tried separating the equations into Real and Imaginary parts but I was not able to decouple the energy terms (real and imaginary)

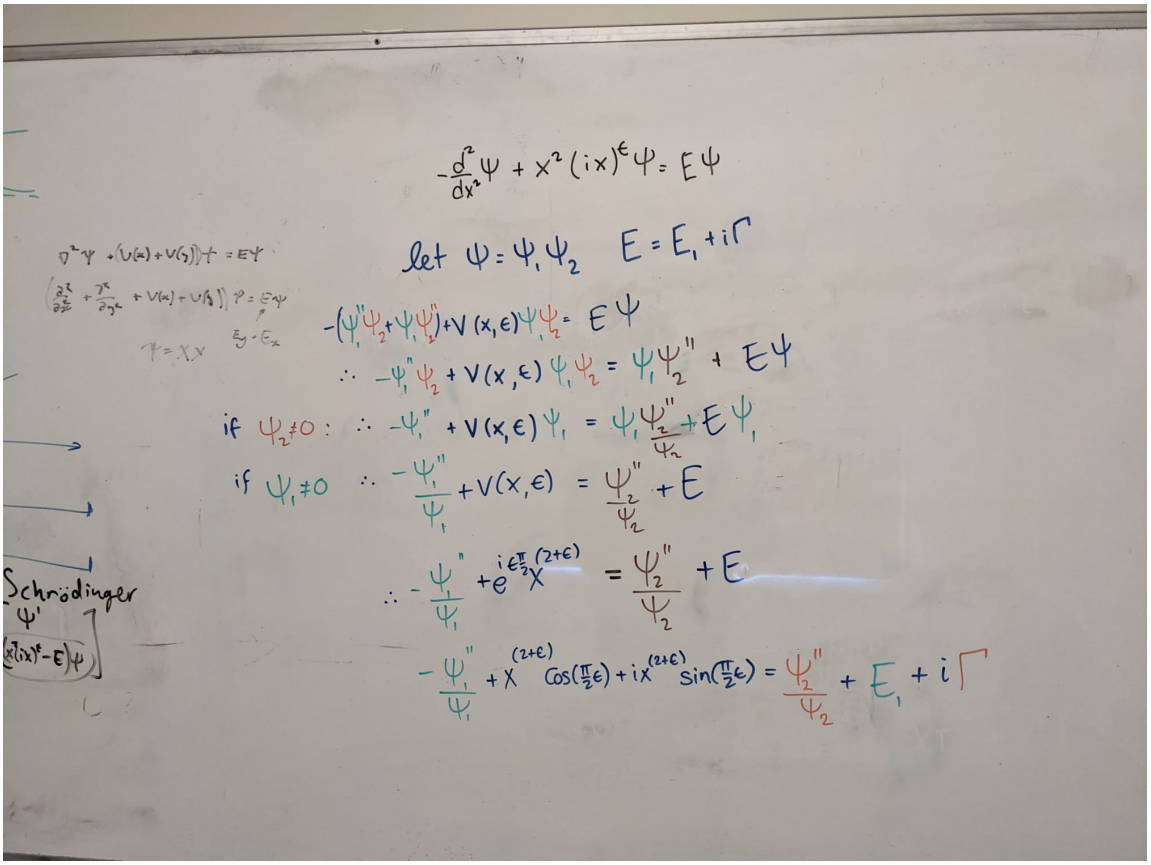


I went through this a couple of times and always got the same result.
Maybe this method is not actually what I need because I would have expected the E_1 and the Γ values to be completely separate.

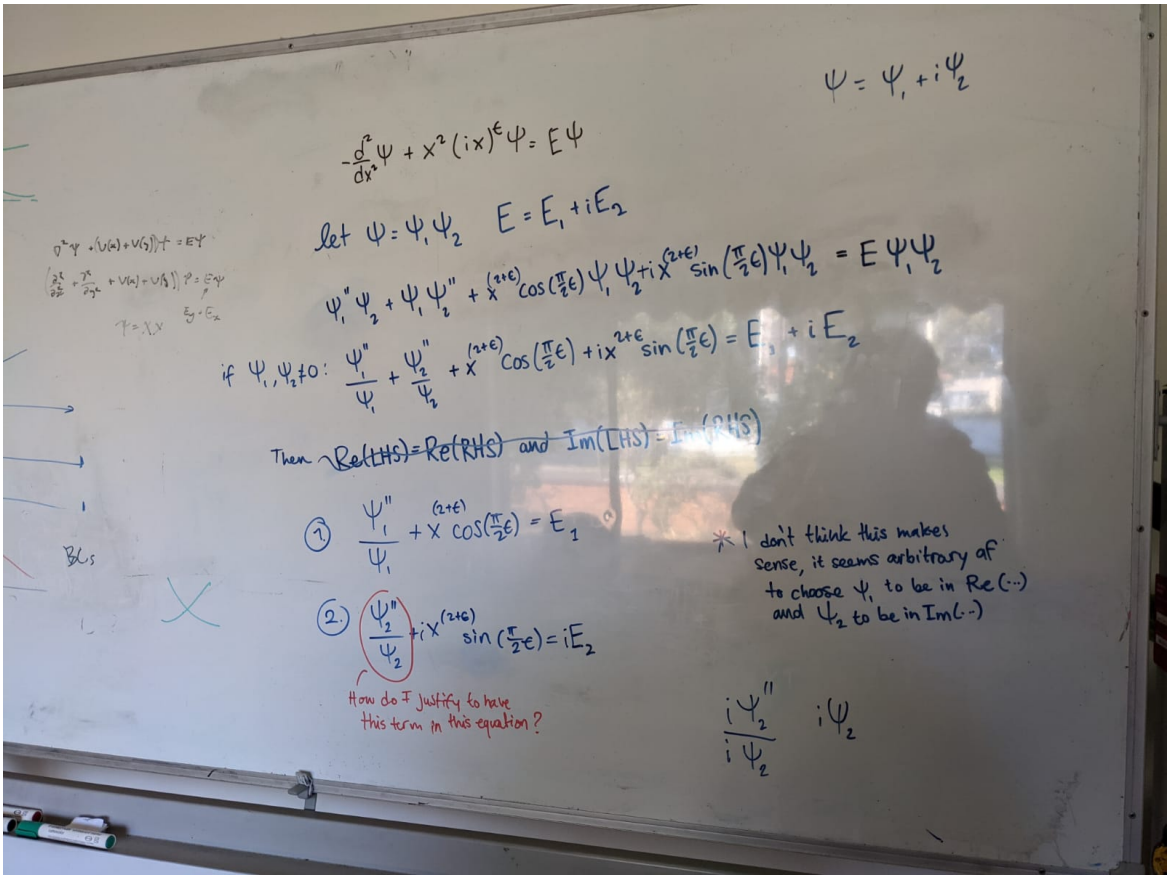
I wonder if I could use some kind alternative version of separation of variables?
This is one separation of variables:



The equations should be set equal to a constant. But then what happens to the energy terms? Do they absorb the constant?



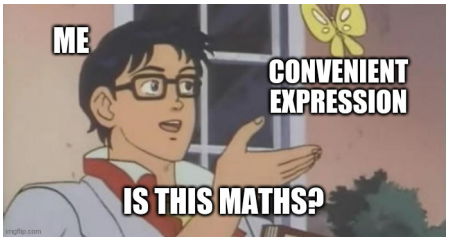
30/04/21



Could I simply separate my equations based on the correspondence between ψ_1, E_1 and ψ_2 and E_2?

Without having to separate them based on realness and imaginaryness?

In the last two lines of this whiteboard work I have 2 solutions that if added could perhaps provide a solution to the original equation.



01/05/21

I asked J&M about my current issue and Jesper replied with this:
"...maybe think about it as a matrix equation: {{H11, H12},{H21,H22}}{psi1,psi2}-Gamma{{0,-1},{1,0}}{psi1,psi2}=E{psi1,psi2}. Now E is an eigenvalue. If you do the expansion in terms of N harmonic oscillator functions, then you will have a 2N x 2N matrix equation that needs to be solved. You will probably need to choose a value of both epsilon and Gamma and solve for E..."

Notes from Ashida

Hermitian Matrices (M) satisfy **NORMALITY**:

$$M^\dagger M = MM^\dagger \Leftrightarrow M = VDV^\dagger$$

with D and V being the diagonal and unitary matrix, respectively.
(IF a matrix is normal THEN it's eigenvectors can be chosen to be orthogonal to each other)

Eigenvectors of non-Hermitian matrices

To generalize orthogonality to non-Hermitian (or more precisely, non-normal) matrices, we should require v_1 to be an eigenvector of $M^\dagger$ with eigenvalue λ_1^* i.e.,

$$M^\dagger \mathbf{v}_1 = \lambda_1^* \mathbf{v}_1 \quad \Leftrightarrow \quad \mathbf{v}_1^\dagger M = \lambda_1 \mathbf{v}_1^\dagger,$$

So that we have

$$\mathbf{v}_1^\dagger M \mathbf{v}_2 = (M^\dagger \mathbf{v}_1)^\dagger \mathbf{v}_2 = \lambda_1 \mathbf{v}_1^\dagger \mathbf{v}_2.$$

We call the eigenvector v_1 of $M^\dagger$ a **left** eigenvector of M. while a conventional eigenvector of M is a **right** eigenvector.

A notable feature of non-Hermitian matrices is that their eigenvectors themselves are generally non-orthogonal, ie. $\langle r_{j'} | r_j \rangle \neq \delta_{j'j}$

Bi-orthogonality - refers to the generalisation of Hermitian orthogonality to non-Hermitian matrices.
Specifically, there is only an orthogonal relation between **left** and **right** eigenvectors of a non-normal (non-normal) matrix M.

BI-ORTHOGONALITY?

Notes from Moiseyev

Unlike the standard situation where the eigenvectors of an Hermitian operator (matrix) form a **complete set** which can be used to expand a wave packet describing the system at a given time, in NHQM it might happen that the eigenfunctions make up an **incomplete set** since several (usually two) eigenvectors coalesce to generate a **self-orthogonal** state (Moiseyev 174).

The c-product

One of the basic postulates of quantum mechanics is that a solution of the time-dependent Schrödinger equation describes the studied system at any given time and so any measurable dynamical quantity can be evaluated from $|\psi(t)\rangle$.
A measurable dynamical property O, is represented by the corresponding operator $\hat{O}$.
In the Hermitian formalism of quantum mechanics:
The projection of $|\psi(t)\rangle$ on a specific eigenstate of an operator$\hat{O}$, denoted by $|n\rangle$, provides the information on the **probability of measuring** in a specific experiment **the corresponding eigenvalue of $\hat{O}$**, labelled by O_n.
The projection of $|\psi(t)\rangle$ on $|n\rangle$ is carried out by using the **scalar product** between any two square integrable analytical functions.

The mean value of the quantity $\bar{O}$ that will be measured in a series of identical experiments is given by the expectation value,

$$\bar{O} = \frac{\langle \Psi(t) | \hat{O} | \Psi(t) \rangle}{\langle \Psi(t) | \Psi(t) \rangle}$$

In order to differentiate between the conventional scalar product $\langle \cdot\cdot | \cdot\cdot \rangle$ Moiseyev labels the functional defined below as the **c-product** by $(\cdot\cdot | \cdot\cdot)$.
Let us associate the eigenstates of the non-Hermitian operator $\hat{O}_\theta$ as $|\psi^\theta{}_n)$with the eigenfunctions of the differential eigenvalue problem:

$$\hat{O}_\theta |\psi^\theta{}_n) = \lambda^\nu{}_n |\psi^\theta{}_n).$$

It's required that the c-product satisfies the orthogonality condition for the eigenstates of $\hat{O}_\theta$

$$(\psi^\theta{}_{n'} | \psi^\theta{}_n) = \delta_{n'n}$$

With the c-product defined the continuous eigenvalue problem can be discretized

$$O_\theta C^\theta{}_n = \lambda^\theta{}_n C^\theta{}_{n'},$$
$$[D^\theta{}_m]^T O_\theta = \lambda^\theta{}_m [D^\theta{}_m]^T,$$
$$[D^\theta{}_{m\neq n}]^T C^\theta{}_n = 0,$$

Where $C^\theta{}_n$ and $[D^\theta{}_m]^T$ are the right and left eigenvectors of the matrix O_θ

Week 9 (03/05/21 – 09/05/21)

Aims:

1. Reproduce Fig 1 in Bender
2. ~~Understand how is it that complex eigenstates of NH systems are not orthogonal~~
    a. ~~Maybe something to do with Bi-orthogonality?~~
3. Simulate a RF-CAP
    a. Omg getting to this is taking me forever :'(

Tasks:

1. **PLOT THE BROKEN SYMMETRY SECTOR OF THE SPECTRUM IN FIG 1.**
    a. **HOW TO SEPARATE MY EQUATION INTO TWO COUPLED EQUATIONS?**
        i. ~~Separation of variables?~~
        ii. maybe think about it as a matrix equation: {{H11, H12},{H21,H22}}{psi1,psi2}-Gamma{{0,-1},{1,0}}{psi1,psi2}=E{psi1,psi2}. Now E is an eigenvalue. If you do the expansion in terms of N harmonic oscillator functions, then you will have a 2N x 2N matrix equation that needs to be solved. You will probably need to choose a value of both epsilon and Gamma and solve for E.
2. Continue Gao's Hamiltonian analysis (using Benders methodology)
3. ~~FINISH progress report Due: Friday 7th May (week 9)~~

Solving a matrix equation

01/05/21
I asked J&M about my current issue and Jesper replied with this:
"...maybe think about it as a matrix equation: {{H11, H12},{H21,H22}}{psi1,psi2}-Gamma{{0,-1},{1,0}}{psi1,psi2}=E{psi1,psi2}. Now E is an eigenvalue. If you do the expansion in terms of N harmonic oscillator functions, then you will have a 2N x 2N matrix equation that needs to be solved. You will probably need to choose a value of both epsilon and Gamma and solve for E..."

03/05/21
Since I need to use harmonic oscillator eigenstates I should refresh my memory as to how to write their general form:
The TISE of the 1D harmonic oscillator is

$$H|n\rangle = E|n\rangle$$
$$(\frac{p^2}{2m} + m^2\omega^2x^2)|n\rangle = (n+\frac{1}{2})\hbar\omega|n\rangle$$
$$(N+\frac{1}{2})\hbar\omega|n\rangle = E_n|n\rangle$$

Where
$N = a^\dagger a$ is the number operator;
$a^\dagger = \frac{1}{\sqrt{2}}(\frac{x}{l_0} - \frac{ipl_0}{\hbar})$ and $a = \frac{1}{\sqrt{2}}(\frac{x}{l_0} + \frac{ipl_0}{\hbar})$ are the creation and annihilation operators respectively
And $l_0 = \sqrt{\frac{\hbar}{m\omega}}$ is the harmonic oscillator length

The states are written as $|n\rangle = \frac{(a^\dagger)}{\sqrt{n!}}|0\rangle$
How do I write the general form of the matrix elements?

If I do the expansion in terms of N harmonic oscillator eigenfunctions

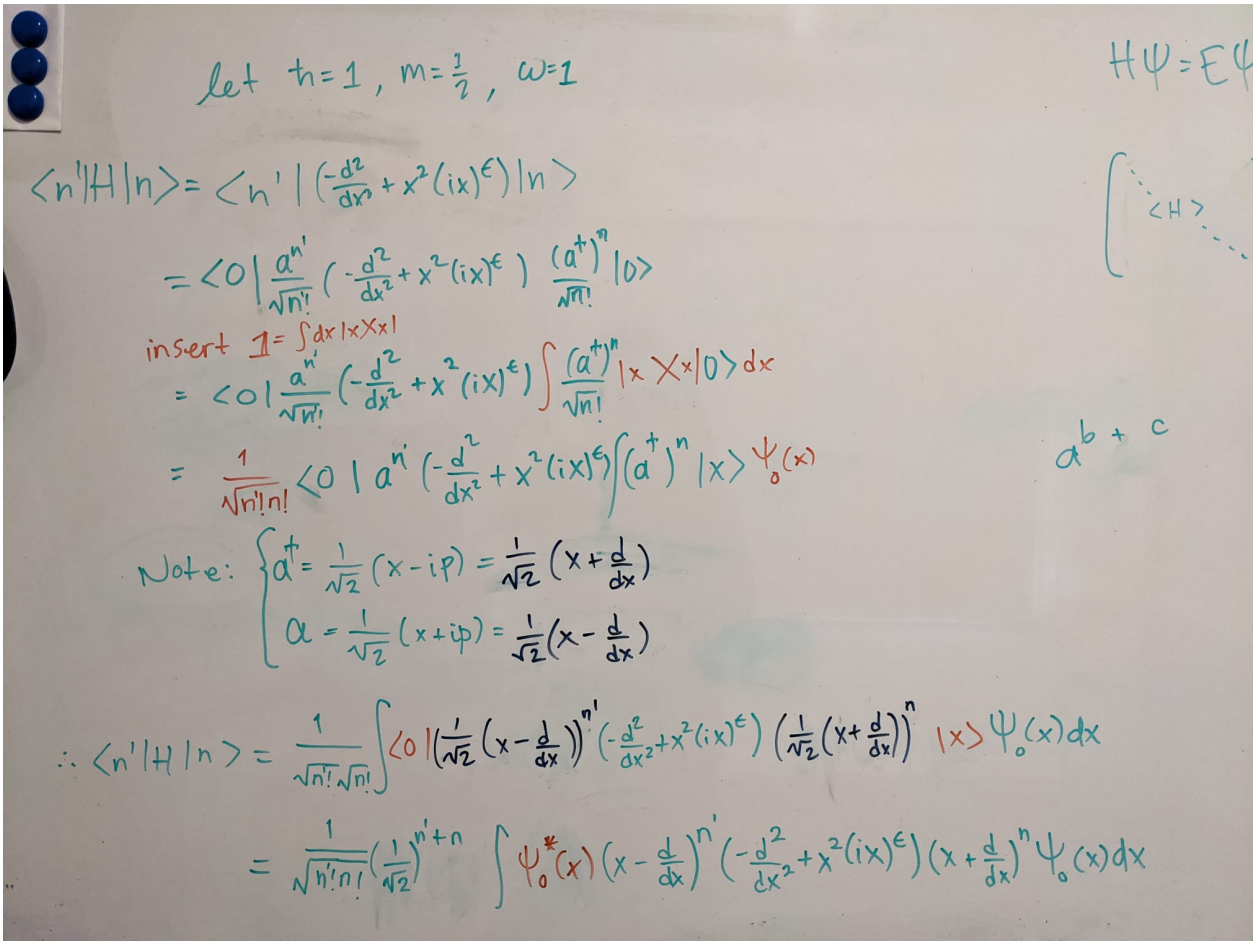
$$|\psi\rangle = \sum_{n=1}^{N} C_n|n\rangle$$

Written in the spatial basis:

$$0 = \langle x|a|0\rangle = \langle x|\frac{1}{\sqrt{2}}(\frac{x}{l_0} + \frac{ipl_0}{\hbar})|0\rangle = \frac{1}{\sqrt{2}}(\frac{x}{l_0} + l_0\frac{\partial}{\partial x})\psi_0(x)$$

Where $\psi_0(x) = \frac{1}{\pi^{1/4}\sqrt{l_0}}e^{-x^2/(2l_0^2)}$
Then the matrix elements of my complex hamiltonian matrix can be written as

$$\langle n'|H|n\rangle = \langle 0|\frac{a^{n'}}{\sqrt{n'!}}\left(-\frac{d^2}{dx^2} + x^2(ix)^\epsilon\right)\frac{(a^\dagger)^n}{\sqrt{n!}}|0\rangle$$



Not sure if this is the best way to do things yet.
Then I have to find the elements iteratively (n=1 → N) by using Mathematica (analytically) or Python (numerically).

If I am right then the matrix equation should look like this:

$$\begin{bmatrix} \ddots & & \\ & \langle n'|H|n\rangle & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ C_n \\ \vdots \end{bmatrix} = (E + i\Gamma) \begin{bmatrix} \vdots \\ C_n \\ \vdots \end{bmatrix}$$

Didn't I need a 2N x 2N matrix equation (according to Jesper) ??? yes… but I think he was talking about a slightly different setup which separated imaginary and real parts but I don't think that I need to separate them per se with my method.

Also, I am still unsure about what the value of Γshould be.
Jesper suggested that first I should try the case forΓ=0 so I get the ϵ=0 solutions, as well as all positive values.
Then take ϵ slightly negative and Γsmall

04/05/21
from wikipedia https://en.wikipedia.org/wiki/Quantum_harmonic_oscillator:

> One may solve the differential equation representing this eigenvalue problem in the coordinate basis, for the wave function $\langle x|\psi\rangle = \psi(x)$, using a spectral method. It turns out that there is a family of solutions. In this basis, they amount to Hermite functions,
>
> $$\psi_n(x) = \frac{1}{\sqrt{2^n\, n!}} \cdot \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \cdot e^{-\frac{m\omega x^2}{2\hbar}} \cdot H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right), \qquad n = 0, 1, 2, \ldots.$$
>
> The functions H_n are the physicists' Hermite polynomials,
>
> $$H_n(z) = (-1)^n\, e^{z^2} \frac{d^n}{dz^n}\left(e^{-z^2}\right).$$
>
> The corresponding energy levels are
>
> $$E_n = \hbar\omega\left(n + \tfrac{1}{2}\right) = (2n+1)\frac{\hbar}{2}\omega\,.$$

I am working with natural length and energy scales therefore in my equations I set
$\hbar = 1,\ m = \frac{1}{2},\ \omega = 1$
Therefore the H.O. eigenstates are:

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!}} \left(\frac{1}{2\pi}\right)^{1/4} e^{\frac{-x^2}{4}} H\left(\frac{1}{\sqrt{2}}x\right)$$

Previously, I worked out a coordinate space expression for my matrix elements:

$$\langle n'|H|n\rangle = \frac{1}{\sqrt{n'!n!}}\left(\frac{1}{\sqrt{2}}\right)^{n'+n}\int \psi_0^*(x)\left(x-\frac{d}{dx}\right)^{n'}\left(-\frac{d^2}{dx^2}+x^2(ix)^\epsilon\right)\left(x+\frac{d}{dx}\right)^{n}\psi_0(x)dx$$

If I write this in terms of $|n\rangle$ without using the creation and destruction operators I will be able to use the harmonic oscillator states as they are in wikipedia.

$$\langle n'|H|n\rangle = \int \psi_{n'}^*(x)\left(-\frac{d^2}{dx^2}+x^2(ix)^\epsilon\right)\psi_n(x)dx$$

This expression looks a lot simpler to code! Yay!

Let me take a step back…

WHAT AM I ACTUALLY DOING?

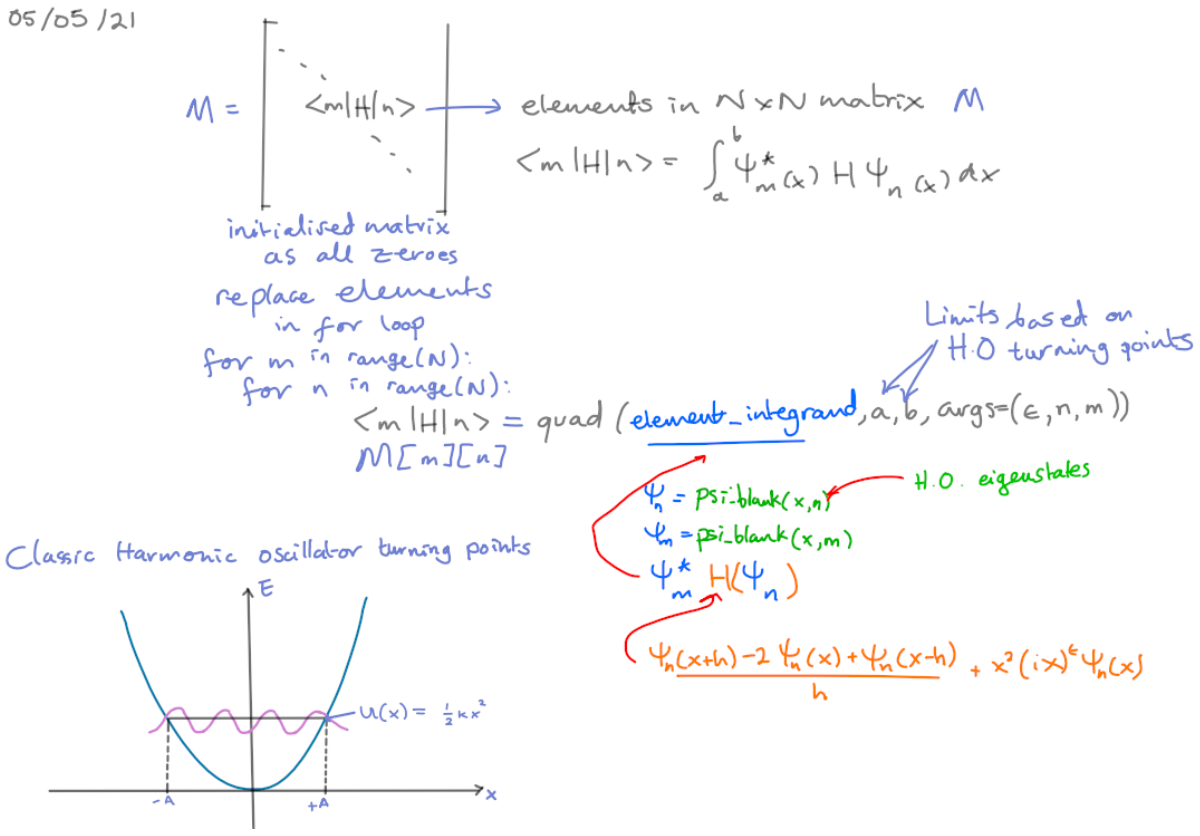
1. Need to find the general form of the H.O. basis functions ✅
    a. Write these in the coordinate basis ✅
2. Decide how I am solving the equation ✅
    a. Numerically?
    b. Analytically?
3. How many functions am I using? [ ]
    a. n→N
4. Make the Hamiltonian matrix [ ]
5. Diagonalise the Hamiltonian matrix [ ]

If I try to solve the equation numerically i need to find:

1. A module in numpy for Hermite polynomials ✅
    a. scipy.special.eval_hermite

05/05/21
MEETING DAY!

Code



```python
################################## Matrix SOLVING ######################################

def psi_blank(x, n):
    return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)

def Hamiltonian(x, ϵ, n):
    h = 1e-6
    d2Ψdx2 = (psi_blank(x + h, n) - 2 * psi_blank(x, n) + psi_blank(x - h, n)) / h**2
    return d2Ψdx2 + (x**2 * (1j * x)**ϵ) * psi_blank(x, n)

def element_integrand(x, ϵ, m, n):
    #CHECK THESE IF mass = 1 instead of 1/2
    psi_m = psi_blank(x, m)
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)

# NxN MATRIX
def Matrix(x, N):
    b = np.abs(1/2 * k * x**2)
    a = - b
    M = np.zeros((N, N))
    for m in range(N):
        for n in range(N):
            element =  quad(element_integrand, a, b, args=(ϵ, m, n))[0]
            # print(element)
            M[m][n] = element
    return M

################################function calls#########################################
# GLOBALS
ϵ = 1
k = 1
x = 2
N = 5

# NxN MATRIX
print(Matrix(x, N))

```

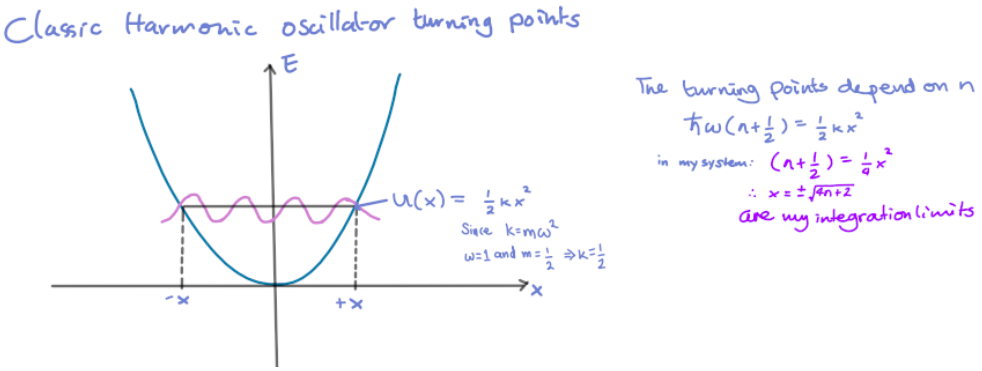
```
[ana:Bender]$ python3 Bending.py
/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py:453: ComplexWarning: Casting complex values to real discards the imaginary part
  return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
Bending.py:55: IntegrationWarning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  element =  quad(element_integrand, a, b, args=(ϵ, m, n))[0]
Bending.py:55: IntegrationWarning: The occurrence of roundoff error is detected, which prevents
  the requested tolerance from being achieved.  The error may be
  underestimated.
  element =  quad(element_integrand, a, b, args=(ϵ, m, n))[0]
[[-0.29262546  0.          0.17329566  0.          0.00337444]
 [ 0.         -0.38490802  0.          0.11814638  0.        ]
 [ 0.06531956  0.         -0.10864931  0.          0.02531868]
 [ 0.          0.04615693  0.         -0.02092683  0.        ]
 [-0.00562369  0.          0.0129458   0.         -0.00375323]]
[ana:Bender]$
```

I need to figure out the IntegrationWarning error displayed in the terminal output above.
Is my Integral diverging? The error remains if I change the limit upper bound on the number of subintervals used in the adaptive algorithm. The error disappears if I decrease the absolute error tolerance from the default value epsabs=1.49e-8 to epsabs=1.49e-04.

```python
################################function calls#########################################
# GLOBALS
ϵ = 1
k = 1
x = 2
N = 5

# NxN MATRIX
Matrix = Matrix(x, N)
print(f"\nMatrix\n{Matrix}")

eigenvalues, eigenvectors = linalg.eig(Matrix)
print(f"\nEigenvalues\n{np.real(eigenvalues)}")
print(f"\nEigenvectors\n{eigenvectors.round(10)}\n")
```

```
[ana:Bender]$ python3 Bending.py
/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py:453: ComplexWarning: Casting complex values to real discards the imaginary part
  return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)

Matrix
[[-0.2925872   0.          0.17335128  0.          0.00337151]
 [ 0.         -0.38492001  0.          0.11815238  0.        ]
 [ 0.06532697  0.         -0.10864738  0.          0.0253203 ]
 [ 0.          0.04616082  0.         -0.02092749  0.        ]
 [-0.00562471  0.          0.01294509  0.         -0.00375349]]

Eigenvalues
[-3.41621207e-01 -6.31439775e-02 -2.22888948e-04 -3.99333146e-01
 -6.51435382e-03]

Eigenvectors
[[-0.96177466  0.59764043 -0.21353037 -0.         -0.        ]
 [ 0.          0.         -0.         -0.99264153  0.29804658]
 [ 0.27256131  0.79328332 -0.34233234  0.         -0.        ]
 [-0.          -0.         -0.          0.12109002  0.95455133]
 [-0.02645424 -0.11630777 -0.91499363  0.         -0.        ]]
```

The only remaining error probably comes from the complexity of the Hamiltonian.
I spoke to Chris and he reminded me that I have to consider the turning points as functions of n
Since they change for each eigenstate of the harmonic oscillator.



Another important factor is that I need to choose the largest integration region and so this depends on the state with the highest energy value.
Ie. if m>n then use x = $\pm\sqrt{4m+2}$ as the integration bounds.

The error I keep getting above
ComplexWarning: Casting complex values to real discards the imaginary part
return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
will likely be solved by using the complex_quad() function I wrote earlier instead of the simple quad()

This is how my matrix function has changed so far:

```
# NxN MATRIX
def Matrix(x, N):
    M = np.zeros((N, N), dtype="complex")
    for m in range(N):
        for n in range(N):
            b = np.abs(np.sqrt(4 * max(m, n) +2))
            element =  complex_quad(element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-04, limit=50)
            # print(element)
            M[m][n] = element
    return M
```

Also...
I added this to the top of my code to set the options that determine the way floating point numbers, arrays and other NumPy objects are displayed.
np.set_printoptions(linewidth=150)
In addition I've noticed that I am printing only the real part of the eigenvalues so… that needed to be corrected since I shouldn't split these.

This is my function call and the output

```
#####################################function calls#############################################
# GLOBALS
ϵ = 1
k = 1/2
x = 2
N = 5

# NxN MATRIX
Matrix = Matrix(x, N)
print(f"\nMatrix\n{Matrix}")

eigenvalues, eigenvectors = linalg.eig(Matrix)
print(f"\nEigenvalues\n{eigenvalues}")
print(f"\nEigenvectors\n{eigenvectors.round(10)}\n")
```

```
[ana:Bender]$ python3 Bending.py

Matrix
[[-3.14431845e-01+0.j          0.00000000e+00+1.47173192j  1.22678063e-01+0.j          0.00000000e+00+0.31659318j -3.01580589e-05+0.j        ]
 [ 0.00000000e+00+1.47173192j -4.06130135e-01+0.j          0.00000000e+00+1.66118894j  6.15235205e-02+0.j          0.00000000e+00+0.15731329j]
 [ 1.14174689e-01+0.j          0.00000000e+00+1.66118894j -1.63978238e-01+0.j          0.00000000e+00+0.64945176j  1.54106505e-02+0.j        ]
 [ 0.00000000e+00+0.31659318j  5.83481016e-02+0.j          0.00000000e+00+0.64945176j -3.77422663e-02+0.j          0.00000000e+00+0.14759278j]
 [-2.99313387e-04+0.j          0.00000000e+00+0.15731329j  1.47804199e-02+0.j          0.00000000e+00+0.14759278j -6.01979143e-03+0.j        ]]

Eigenvalues
[-0.22216423-2.32564960e+00j -0.22216423+2.32564960e+00j -0.20696033-1.27228238e-01j -0.20696033+1.27228238e-01j -0.07005317+1.24769119e-17j]

Eigenvectors
[[-0.45261993-0.01730565j  0.45261993-0.01730565j  0.31563889-0.33597708j -0.31563889-0.33597708j  0.16781635-0.j        ]
 [ 0.67287772+0.j          0.67287772+0.j         -0.22722411-0.01482775j -0.22722411+0.01482775j -0.        -0.15480592j]
 [-0.53745901-0.05022901j  0.53745901-0.05022901j -0.24984559+0.37432626j  0.24984559+0.37432626j -0.27360699+0.j        ]
 [ 0.2113268 +0.05077916j  0.2113268 -0.05077916j  0.67031683+0.j          0.67031683+0.j          0.        +0.48400409j]
 [-0.0575015 -0.01192426j  0.0575015 -0.01192426j -0.14963143-0.24775722j  0.14963143-0.24775722j  0.79921966+0.j        ]]
```

Yay, now my matrix is complex valued.

Wowwowowow hold your horses friend.
I need to actually make sure the states have gone to zero in order to take advantage of using the H.O. states.
The limits of integration must be much larger than the turning points of the H.O. so I can guarantee that the states have exponentially decayed to zero!

```python
# NxN MATRIX
def Matrix(x, N):
    M = np.zeros((N, N), dtype="complex")
    for m in range(N):
        for n in range(N):
            b = 5 * np.abs(np.sqrt(4 * max(m, n) +2))
            element =  complex_quad(element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-04, limit=50)
            # print(element)
            M[m][n] = element
    return M
```

The integration limit b is now b = 5 * np.abs(np.sqrt(4 * max(m, n) +2))
The function calls are done with the exact same parameters as my previous call.
This is the new output:

```
[ana:Bender]$ python3 Bending.py

Matrix
[[-3.14431845e-01+0.j          0.00000000e+00+1.47173192j  1.22678063e-01+0.j          0.00000000e+00+0.31659318j -3.01580589e-05+0.j        ]
 [ 0.00000000e+00+1.47173192j -4.06130135e-01+0.j          0.00000000e+00+1.66118894j  6.15235205e-02+0.j          0.00000000e+00+0.15731329j]
 [ 1.14174689e-01+0.j          0.00000000e+00+1.66118894j -1.63978238e-01+0.j          0.00000000e+00+0.64945176j  1.54106505e-02+0.j        ]
 [ 0.00000000e+00+0.31659318j  5.83481016e-02+0.j          0.00000000e+00+0.64945176j -3.77422663e-02+0.j          0.00000000e+00+0.14759278j]
 [-2.99313387e-04+0.j          0.00000000e+00+0.15731329j  1.47804199e-02+0.j          0.00000000e+00+0.14759278j -6.01979143e-03+0.j        ]]

Eigenvalues
[-0.22216423-2.32564960e+00j -0.22216423+2.32564960e+00j -0.20696033-1.27228238e-01j -0.20696033+1.27228238e-01j -0.07005317+1.24769119e-17j]

Eigenvectors
[[-0.45261993-0.01730565j  0.45261993-0.01730565j  0.31563889-0.33597708j -0.31563889-0.33597708j  0.16781635-0.j        ]
 [ 0.67287772+0.j          0.67287772+0.j         -0.22722411-0.01482775j -0.22722411+0.01482775j -0.        -0.15480592j]
 [-0.53745901-0.05022901j  0.53745901-0.05022901j -0.24984559+0.37432626j  0.24984559+0.37432626j -0.27360699+0.j        ]
 [ 0.2113268 +0.05077916j  0.2113268 -0.05077916j  0.67031683+0.j          0.67031683+0.j          0.        +0.48400409j]
 [-0.0575015 -0.01192426j  0.0575015 -0.01192426j -0.14963143-0.24775722j  0.14963143-0.24775722j  0.79921966+0.j        ]]
```

It looks the same as before..
What if I increase b to be b = 10 * np.abs(np.sqrt(4 * max(m, n) +2))

```
[ana:Bender]$ python3 Bending.py

Matrix
[[-2.50006445e-01+0.j          0.00000000e+00+2.12132034j  1.25018028e-01+0.j          0.00000000e+00+0.35355339j  4.05330418e-07+0.j        ]
 [ 0.00000000e+00+2.12132034j -3.74984626e-01+0.j          0.00000000e+00+2.12132034j  6.25088167e-02+0.j          0.00000000e+00+0.1767767j ]
 [ 1.25010662e-01+0.j          0.00000000e+00+2.12132034j -1.56254714e-01+0.j          0.00000000e+00+0.79549513j  1.56204853e-02+0.j        ]
 [ 0.00000000e+00+0.35355339j  6.25136818e-02+0.j          0.00000000e+00+0.79549513j -3.64571517e-02+0.j          0.00000000e+00+0.1767767j ]
 [-3.10994755e-06+0.j          0.00000000e+00+0.1767767j   1.56229638e-02+0.j          0.00000000e+00+0.1767767j  -5.86031867e-03+0.j        ]]

Eigenvalues
[-0.19631852+3.11060875e+00j -0.19631852-3.11060875e+00j -0.18789161+2.92510149e-01j -0.18789161-2.92510149e-01j -0.05514299+1.03527829e-17j]

Eigenvectors
[[ 0.48459202-0.01610338j -0.48459202-0.01610338j -0.44054282-0.17305833j  0.44054282-0.17305833j  0.18356763-0.j        ]
 [ 0.68111613+0.j          0.68111613+0.j         -0.19240555+0.01244457j -0.19240555-0.01244457j -0.        -0.08450738j]
 [ 0.51112629-0.03513314j -0.51112629-0.03513314j  0.39718998+0.22408344j -0.39718998+0.22408344j -0.27927918+0.j        ]
 [ 0.18678478-0.03452957j  0.18678478+0.03452957j  0.68031215+0.j          0.68031215+0.j          0.        +0.30711636j]
 [ 0.04868677-0.00750998j -0.04868677-0.00750998j  0.21503378-0.14751495j -0.21503378-0.14751495j  0.88704234+0.j        ]]
```

Cool, so things have changed a fair amount now.
If I change by increasing it to b = 15 * np.abs(np.sqrt(4 * max(m, n) +2))

```
[ana:Bender]$ python3 Bending.py

Matrix
[[-2.49988703e-01+0.j          0.00000000e+00+2.12132034j  1.24986420e-01+0.j          0.00000000e+00+0.35355339j  5.45562192e-06+0.j        ]
 [ 0.00000000e+00+2.12132034j -3.74965362e-01+0.j          0.00000000e+00+2.12132034j  6.24707623e-02+0.j          0.00000000e+00+0.1767767j ]
 [ 1.24995412e-01+0.j          0.00000000e+00+2.12132034j -1.56266940e-01+0.j          0.00000000e+00+0.79549513j  1.56263235e-02+0.j        ]
 [ 0.00000000e+00+0.35355339j  6.24955289e-02+0.j          0.00000000e+00+0.79549513j -3.64533292e-02+0.j          0.00000000e+00+0.1767767j ]
 [ 3.44916911e-07+0.j          0.00000000e+00+0.1767767j   1.56232246e-02+0.j          0.00000000e+00+0.1767767j  -5.86025724e-03+0.j        ]]
[ana:Bender]$
```

The matrix elements do not change that much anymore..
Let's leave this for now as b = 10 * np.abs(np.sqrt(4 * max(m, n) +2))

If I increase N from the test value N = 5 then I will get more resolution in Harmonic oscillator space.
The more eigenstates I use the more degrees of freedom I have which means the more accurate I am in my calculation of the eigenenergies of the non-Hermitian Hamiltonian.

The thing is that if I increase the number of states used to make my matrix the number of eigenvalues will increase linearly with N.
I only want to have the first 10 or so eigenvalues :/
In addition the eigenvalues in my output are mixed up in a real and complex eigenvalues list.

So I will have to separate them and sort the real ones in order with increasing energy. To later plot only the lowest ones.

First I want to run my code with N = 100 and check the running time

My base case is the case N = 5

```
[ana:Bender]$ time python3 Bending.py

Matrix
[[-2.50006445e-01+0.j          0.00000000e+00+2.12132034j  1.25018028e-01+0.j          0.00000000e+00+0.35355339j  4.05330418e-07+0.j        ]
 [ 0.00000000e+00+2.12132034j -3.74984626e-01+0.j          0.00000000e+00+2.12132034j  6.25088167e-02+0.j          0.00000000e+00+0.1767767j ]
 [ 1.25010662e-01+0.j          0.00000000e+00+2.12132034j -1.56254714e-01+0.j          0.00000000e+00+0.79549513j  1.56204853e-02+0.j        ]
 [ 0.00000000e+00+0.35355339j  6.25136818e-02+0.j          0.00000000e+00+0.79549513j -3.64571517e-02+0.j          0.00000000e+00+0.1767767j ]
 [-3.10994755e-06+0.j          0.00000000e+00+0.1767767j   1.56229638e-02+0.j          0.00000000e+00+0.1767767j  -5.86031867e-03+0.j        ]]

Eigenvalues
[-0.19631852+3.11060875e+00j -0.19631852-3.11060875e+00j -0.18789161+2.92510149e-01j -0.18789161-2.92510149e-01j -0.05514299+1.03527829e-17j]

real    0m1.213s
user    0m1.339s
sys     0m0.238s
[ana:Bender]$
```

Testing N = 100

```
Matrix
[[-2.50006445e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j  1.25018028e-001+0.00000000e+000j ...  0.00000000e+000+1.37840261e-091j
  2.35558725e-092+0.00000000e+000j  0.00000000e+000-2.10238263e-093j]
 [ 0.00000000e+000+2.12132034e+000j -3.74984626e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j ...  2.23420771e-091+0.00000000e+000j
  0.00000000e+000-7.01565444e-092j  1.25393579e-092+0.00000000e+000j]
 [ 1.25010662e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j -1.56254714e-001+0.00000000e+000j ...  0.00000000e+000+1.96655981e-091j
  3.43791996e-091+0.00000000e+000j  0.00000000e+000-3.04813680e-093j]
 ...
 [ 0.00000000e+000+1.37840261e-091j  2.74324564e-092+0.00000000e+000j  0.00000000e+000+1.96655981e-091j ... -3.51408157e-180+0.00000000e+000j
  0.00000000e+000+5.95158862e-180j  8.19471954e-183+0.00000000e+000j]
 [-3.47571687e-093+0.00000000e+000j  0.00000000e+000-7.01565444e-092j  3.56142027e-093+0.00000000e+000j ...  0.00000000e+000+5.95158862e-180j
  -1.46993470e-182+0.00000000e+000j  0.00000000e+000+2.43106603e-182j]
 [ 0.00000000e+000-2.10238263e-093j -1.77085089e-094+0.00000000e+000j  0.00000000e+000-3.04813680e-093j ...  8.09068404e-183+0.00000000e+000j
  0.00000000e+000+2.43106603e-182j -9.41864020e-185+0.00000000e+000j]]

Eigenvalues
[-1.96284281e-001-3.11079087e+000j -1.96284281e-001+3.11079087e+000j -1.86806232e-001+2.94237675e-001j -1.86806232e-001-2.94237675e-001j
 -5.08088885e-002-1.02057392e-018j -6.48641816e-003+8.26199228e-019j -7.95423813e-004-5.10437221e-020j -2.33319559e-005-9.55181815e-021j
 -8.72295548e-007-4.85211919e-007j -8.72295548e-007+4.85211919e-007j  1.13618556e-010-4.98327067e-009j  1.13618556e-010+4.98327067e-009j
 -8.58072011e-011-7.52073661e-026j -1.75473190e-012+5.76968664e-027j -5.28356244e-014-1.91133305e-028j -6.11244766e-016+4.91324640e-030j
 -1.13719001e-017-9.14629788e-032j -8.11009892e-019-6.30140204e-033j  9.41292961e-021+2.57691677e-020j  9.41292961e-021-2.57691677e-020j
 -2.69594121e-022+4.88587801e-036j -8.06045504e-025-1.70849802e-038j -4.35698640e-026+2.12319716e-039j -1.73628968e-027-5.15923933e-041j
 -4.40549046e-030-3.47427365e-043j  8.07214202e-032+2.87663416e-046j  7.82578451e-033-3.32183779e-046j -7.78540832e-034+2.55115433e-046j
 -1.31677139e-035-5.98862805e-048j -2.76166585e-037+2.19698377e-049j  1.48891365e-040+3.40243668e-053j  3.45788383e-042+4.54060396e-054j
 -1.82167505e-042+2.59559685e-054j -1.13795069e-044-2.09460990e-057j -9.02179192e-048+8.48426306e-060j -4.47042548e-050-2.01932294e-061j
 -2.36410291e-051+1.72484721e-062j -2.33158849e-052-4.16586670e-052j -2.33158849e-052+4.16586670e-052j -3.14217628e-057+4.22108930e-057j
 -3.14217628e-057-4.22108930e-057j -9.08076551e-061-2.90260857e-072j -3.41494386e-063+1.82874489e-062j -3.41494386e-063-1.82874489e-062j
  1.26042820e-067-8.27800743e-079j -4.03595827e-070+9.42054104e-069j -4.03595827e-070-9.42054104e-069j  3.20101339e-070+2.70101946e-081j
 -4.14337063e-073+8.33160193e-085j -1.74293859e-074-3.84179891e-086j -1.07061534e-076-2.21267334e-088j -2.69943855e-079+1.21343665e-091j
 -6.52182622e-081+5.61643835e-093j  1.33982382e-083-1.58972499e-096j  1.83914208e-085-8.12836651e-098j -2.30099673e-088+1.34674502e-100j
 -8.20680709e-090+1.25359634e-100j -6.60504696e-092+1.39217929e-102j -2.44741625e-092-4.92143510e-103j -7.37217567e-095-1.37131214e-105j
 -1.88658889e-097-4.47840913e-110j  1.65427365e-100+1.35093272e-110j  1.06971273e-101-8.98344019e-112j  1.65772648e-105+9.05153492e-106j
  1.65772648e-105-9.05153492e-106j  3.63952270e-109-6.88661244e-109j  3.63952270e-109+6.88661244e-109j -2.59718410e-113+4.78786357e-113j
 -2.59718410e-113-4.78786357e-113j  1.74236171e-116+1.85993259e-127j  2.14818761e-118-3.75809821e-129j  1.34889035e-120-2.62753459e-131j
  1.43980029e-123-2.69516079e-134j -5.36054599e-128+3.83829617e-127j -5.36054599e-128-3.83829617e-127j  8.48099252e-130+1.27316124e-140j
  1.77262908e-130-2.67759380e-141j -3.46192368e-134+6.12648755e-145j -2.10030289e-135-1.78603228e-147j -5.16434519e-139-5.52359528e-149j
 -8.20508654e-140+5.45743518e-150j -2.23128979e-142+1.75000445e-153j  3.01314918e-146+1.68558752e-156j  3.55907171e-148+1.92181751e-158j
  9.25898966e-149-3.57312779e-159j -3.32292019e-152-1.29428255e-152j -3.32292018e-152+1.29428255e-152j  8.20759993e-156+2.15207950e-166j
 -1.30369556e-158+1.90465107e-168j -1.72964909e-159-3.08548554e-169j  4.43448623e-163-4.63782965e-173j  6.01076846e-165+8.59004536e-175j
  8.58765332e-184-5.35666819e-194j  2.29391786e-167-5.70219495e-178j  2.00187598e-169+1.73334168e-179j -1.27176371e-171+3.15292642e-181j
 -8.27760205e-173-3.47504541e-183j -6.41980640e-176+1.15826121e-185j -3.10537764e-178-1.22236225e-187j -4.80390413e-180+9.59599946e-190j]

real    1m14.856s
user    1m15.031s
sys     0m0.430s
```

It's slow.. But not awful

Testing N = 500

```
[ana:Bender]$ time python3 Bending.py
Bending.py:48: RuntimeWarning: invalid value encountered in double_scalars
  return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
Bending.py:17: IntegrationWarning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  real_integral = quad(real_func, a, b, **kwargs)
Bending.py:18: IntegrationWarning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  imag_integral = quad(imag_func, a, b, **kwargs)
Bending.py:17: IntegrationWarning: The occurrence of roundoff error is detected, which prevents
  the requested tolerance from being achieved.  The error may be
  underestimated.
  real_integral = quad(real_func, a, b, **kwargs)
Bending.py:18: IntegrationWarning: The occurrence of roundoff error is detected, which prevents
  the requested tolerance from being achieved.  The error may be
  underestimated.
  imag_integral = quad(imag_func, a, b, **kwargs)
Bending.py:48: RuntimeWarning: overflow encountered in multiply
  return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
^CTraceback (most recent call last):
  File "Bending.py", line 79, in <module>
    Matrix = Matrix(x, N)
  File "Bending.py", line 66, in Matrix
    element =  complex_quad(element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-04, limit=50)
  File "Bending.py", line 17, in complex_quad
    real_integral = quad(real_func, a, b, **kwargs)
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 341, in quad
    retval = _quad(func, a, b, args, full_output, epsabs, epsrel, limit,
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 453, in _quad
    return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
  File "Bending.py", line 14, in real_func
    return np.real(func(*args))
  File "Bending.py", line 58, in element_integrand
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)
  File "Bending.py", line 53, in Hamiltonian
    return d2Ψdx2 + (x**2 * (1j * x)**ϵ) * psi_blank(x, n)
  File "Bending.py", line 48, in psi_blank
    return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
KeyboardInterrupt


real    4m21.260s
user    4m20.998s
sys     0m0.340s
[ana:Bender]$
```

Okay! This kinda sucks. I wanted to get N to 1000 and it's looking like I have to dissect the way in which the states are written… The problem is probably in

```
def psi_blank(x, n):
        return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
```

There might be an issue with the size of the state given that I am using 2**n and an n factorial in the denominator. I need to verify that sc.eval_hermite(n, np.sqrt(1/2) * x does actually give me something large to scale things adequately for large n.
I need to think about things for a bit.

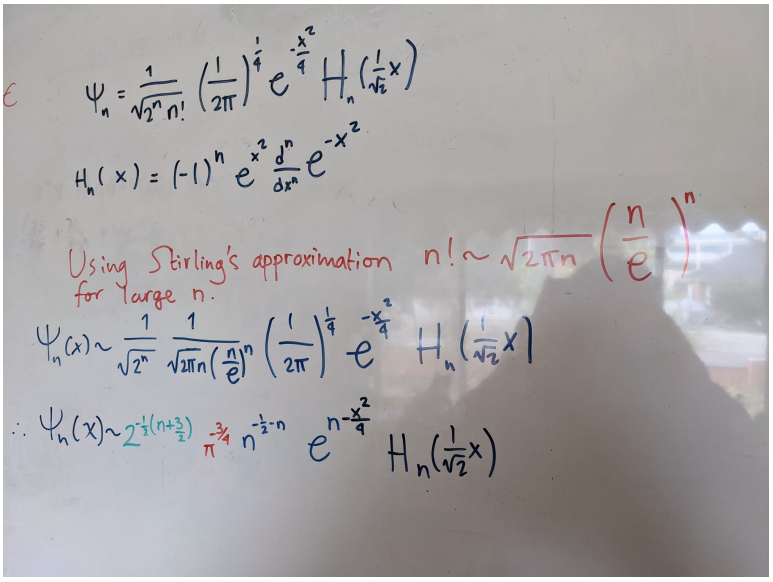
07/05/21
How “large” are Hermite polynomials?

```
print(f"n = 10: {sc.eval_hermite(10, np.sqrt(1/2) * x)}\n")
print(f"n = 100: {sc.eval_hermite(100, np.sqrt(1/2) * x)}\n")
print(f"\nn = 500: {sc.eval_hermite(500, np.sqrt(1/2) * x)}\n")
```

```
[ana:Bender]$ time python3 Bending.py
n = 10: -83872.00000000003

n = 100: 3.283927976363779e+93


n = 500: nan


real    0m0.670s
user    0m0.765s
sys     0m0.247s
```

scipy.special.eval_hermite(*n*, *x*) blows up very fast!

I can try to re-write my basis states for large n using Stirling’s approximation:




```
def psi_blank(x, n):
    if n < 100:
        return (1 / (2**n * sc.factorial(n))) * (1/ (2 * np.pi))**(1/4) * np.exp(-x**2/4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
    else:
        return 2**(-1/2 * ( n + 3/2)) * np.pi**-3/4 * n**(-1/2 - n) * np.exp(n - (x**2) / 4) * sc.eval_hermite(n, np.sqrt(1/2) * x)
```

Testing:
N = 100

```
Matrix
[[-2.50006445e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j  1.25018028e-001+0.00000000e+000j ...  0.00000000e+000+1.37840261e-091j  2.35558725e-092+0.00000000e+000j
  0.00000000e+000-2.10238263e-093j]
 [ 0.00000000e+000+2.12132034e+000j -3.74984626e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j ...  2.23420771e-091+0.00000000e+000j  0.00000000e+000-7.01565444e-092j
  1.25393579e-092+0.00000000e+000j]
 [ 1.25010662e-001+0.00000000e+000j  0.00000000e+000+2.12132034e+000j -1.56254714e-001+0.00000000e+000j ...  0.00000000e+000+1.96655981e-091j  3.43791996e-091+0.00000000e+000j
  0.00000000e+000-3.04813680e-093j]
 ...
 [ 0.00000000e+000+1.37840261e-091j  2.74324564e-092+0.00000000e+000j  0.00000000e+000+1.96655981e-091j ... -3.51408157e-180+0.00000000e+000j  0.00000000e+000+5.95158862e-180j
  8.19471954e-183+0.00000000e+000j]
 [-3.47571687e-093+0.00000000e+000j  0.00000000e+000-7.01565444e-092j  3.56142027e-093+0.00000000e+000j ...  0.00000000e+000+5.95158862e-180j -1.46993470e-182+0.00000000e+000j
  0.00000000e+000+2.43106603e-182j]
 [ 0.00000000e+000-2.10238263e-093j -1.77085089e-094+0.00000000e+000j  0.00000000e+000-3.04813680e-093j ...  8.09068404e-183+0.00000000e+000j  0.00000000e+000+2.43106603e-182j
  -9.41864020e-185+0.00000000e+000j]]

real    1m14.187s
user    1m14.312s
sys     0m0.240s
```

N = 500
Matrix doesn’t render.
Bending.py:52: RuntimeWarning: invalid value encountered in double_scalars
  return 2**(-1/2 * (n + 3/2)) * np.pi**-3/4 * n**(-1/2 - n) * np.exp(n - (x**2) / 4) * sc.eval_hermite(n, np.sqrt(1/2) * x)

How big are the parts used in
psi_blank(x) ~ 2**(-1/2 * (n + 3/2)) * np.pi**-3/4 * n**(-1/2 - n) * np.exp(n - (x**2) / 4) * sc.eval_hermite(n, np.sqrt(1/2) * x)

```
Hermite factor
n = 10: -83872.00000000003

n = 100: 3.283927976363779e+93

n = 500: nan

exponential factor
n = 10: 8103.083927575384

n = 100: 9.889030319346946e+42

n = 500: 5.16352720736287l5e+216

other factor
n = 10: 4.737703911963764e-15

n = 100: 4.2581221013322615e-219

n = 500: 0.0
```

Why is the last result zero? Shouldn't it be at least a very very small number?
AHH it turns out that the computer returns 0.0 when there is an **underflow** error.

While googling around about underflow in this calculation related to my inability to calculate higher order HO basis functions, I found this:
https://scicomp.stackexchange.com/questions/30896/generate-high-n-quantum-harmonic-oscillator-states-numerically

Which lead to this:
https://www.numbercrunch.de/blog/2014/08/calculating-the-hermite-functions/

I've been reading H Bauke's blog post and his work appears to be extremely relevant to my research.
In addition, the post that led me to H Bauke's blog post has been extremely useful.

Calculating the Hermite functions

At high N values, the Hermite polynomials blow up and the normalising factor (**other factor**) becomes so small that underflow occurs.
H Bauke solves this by a recurrence relation definition for the Hermite polynomials:

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$$

Where $H_0(x) = 1$and $H_1(x) = 2x$.
The Hermite functions can now be written using this recurrence relation

$$h_n(x) = \sqrt{\frac{2}{n}}\, x\, h_{n-1}(x) - \sqrt{\frac{n-1}{n}} h_{n-2}(x)$$

Thanks to stackoverflow poster

answered Jan 18 '19 at 12:58
user28077

His response to the original question was what made me more able to understand better the following code:

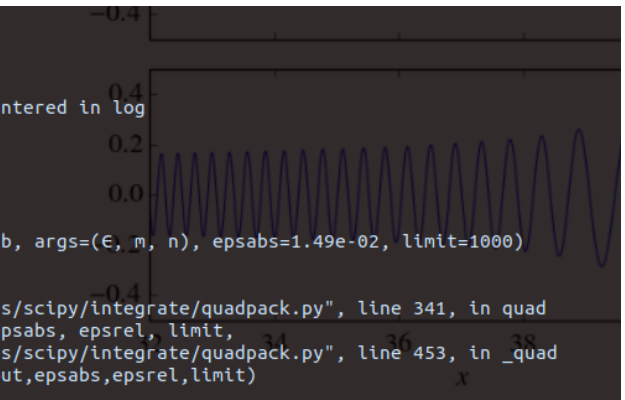
H Bauke's Hermite function:

```
def h(n, x):
    if n==0:
        return ones_like(x)*pi**(-0.25)*exp(-x**2/2)
    if n==1:
        return sqrt(2.)*x*exp(-x**2/2)*pi**(-0.25)*exp(-x**2/2)
    h_i_2=ones_like(x)*pi**(-0.25)
    h_i_1=sqrt(2.)*x*pi**(-0.25)
    sum_log_scale=zeros_like(x)
    for i in range(2, n+1):
        h_i=sqrt(2./i)*x*h_i_1-sqrt((i-1.)/i)*h_i_2
        h_i_2, h_i_1=h_i_1, h_i
        log_scale=log(abs(h_i)).round()
        scale=exp(-log_scale)
        h_i=h_i*scale
        h_i_1=h_i_1*scale
        h_i_2=h_i_2*scale
        sum_log_scale+=log_scale
    return h_i*exp(-x**2/2+sum_log_scale)
```

08/05/21

By using a modified version of Bauke's code partly based on the post by stackoverflow user28077,
I've noticed that the Hermite polynomials I am using in my basis functions **underflow** as x = 0.

```
i=3
h_i_1=-1.4437500620375212
h_i_2=0.0
x=0.0
Bending.py:73: RuntimeWarning: divide by zero encountered in log
  log_scale = np.log(abs(h_i)).round()
Traceback (most recent call last):
  File "Bending.py", line 117, in <module>
    Matrix = Matrix(x, N)
  File "Bending.py", line 104, in Matrix
    element =  complex_quad(element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-02, limit=1000)
  File "Bending.py", line 20, in complex_quad
    real_integral = quad(real_func, a, b, **kwargs)
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 341, in quad
    retval = _quad(func, a, b, args, full_output, epsabs, epsrel, limit,
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 453, in _quad
    return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
  File "Bending.py", line 15, in real_func
    return np.real(func(*args))
  File "Bending.py", line 96, in element_integrand
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)
  File "Bending.py", line 90, in Hamiltonian
    d2Ψdx2 = (psi_blank(x + h, n) - 2 * psi_blank(x, n) + psi_blank(x - h, n)) / h**2
  File "Bending.py", line 74, in psi_blank
    assert h_i != 0
AssertionError
[ana:Bender]$
```



The Hermite function $h_{800}(x)$. A direct calculation via the
Hermite function in terms of the modified Hermite polynon
$x > 39$ due to numerical underflow as shown in the uppe
Python code, which avoids underflow, has been utilized to
part.

I would like to thank Randolf Beerwerth for drawing my atte

I think that this is due to Bauke's "rescaled" recurrence relation definition

```
...       h_i=sqrt(2./i)*x*h_i_1-sqrt((i-1.)/i)*h_i_2
          h_i_2, h_i_1=h_i_1, h_i
          log_scale=log(abs(h_i)).round()
```

It is the log_scale line that presents us with the issue since log(0) is undefined

An pseudo-code outline of what Bauke's code is doing:
Define the Hermite function for some n, and x:

```
    if n is 0:
        Make a numpy object similar to x but with one(s) as entry(ies) multiplied by pi^(-0.25) exp(-x**2/2)
    if n==1:
        The code in this line has a typo!
        return sqrt(2.)*x*exp(-x**2/2)*pi**(-0.25)*exp(-x**2/2)

    The code continues and makes a numpy object similar to x but with one(s) as entry(ies) multiplied by
pi^(-0.25)      and another numpy object also similar to x but scaled by pi^(-0.25) and sqrt 2
    h_i_2=ones_like(x)*pi**(-0.25)
    h_i_1=sqrt(2.)*x*pi**(-0.25)

This numpy object is similar to x and full of zeros. I think this is used to convert the calculation into some
log space calculation somehow. I do not understand this very well.
    sum_log_scale=zeros_like(x)
```

for n > 2:

Define the recurrence relation for the modified Hermite polynomial

$$\tilde{H}_n(x) = \sqrt{\frac{2}{n}}\, x\tilde{H}_{n-1}(x) - \sqrt{\frac{n-1}{n}}\,\tilde{H}_{n-2}(x) \qquad (3)$$

Redefine the modified Hermite polynomials by increasing n
(this makes the newly calculated H_n the new H_{n-1} and the old H_{n-1} into the new H_{n-2})

```
Take the log of the absolute value of H_n
log_scale=log(abs(h_i)).round()

Exponentiate the negative of log_scale
 scale=exp(-log_scale)

To create a scaling factor for all modified Hermite polynomials
 h_i=h_i*scale
 h_i_1=h_i_1*scale
 h_i_2=h_i_2*scale

I don't understand this line well enough, I guess this is Bauke's solution to the underflow issue.
 sum_log_scale+=log_scale

Return function and multiply by the gaussian factor
return h_i*exp(-x**2/2+sum_log_scale)
```

My version of Bauke's hermite function:

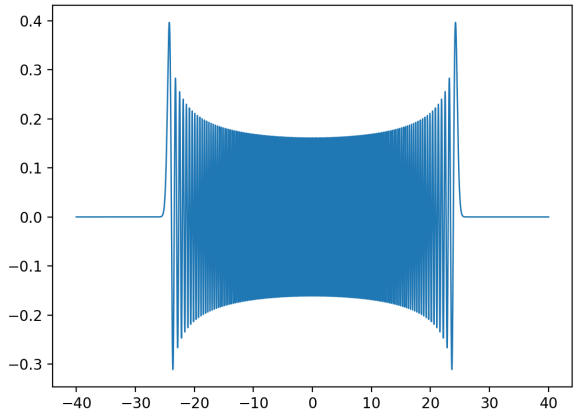
```
def psi_blank(x, n):

    pi_powered = np.pi ** (-0.25)

    x = np.array(x)
    if n == 0:
        return np.ones_like(x) * pi_powered * np.exp(-x ** 2 / 2)
    if n == 1:
        return (
            np.sqrt(2.0)
            * x
            * np.pi ** (-0.25)
            * np.exp(-x ** 2 / 2)
        )
    h_i_2 = np.ones_like(x) * pi_powered
    h_i_1 = np.sqrt(2.0) * x * pi_powered
    sum_log_scale = np.zeros_like(x)
    for i in range(2, n + 1):

        # RECURRENCE RELATION
        h_i = np.sqrt(2.0 / i) * x * h_i_1 - np.sqrt((i - 1.0) / i) * h_i_2
        h_i_2, h_i_1 = h_i_1, h_i

        x[x == 0] = 1e-200

        log_scale = np.log(abs(h_i)).round()
        scale = np.exp(-log_scale)
        h_i = h_i * scale
        h_i_1 = h_i_1 * scale
        h_i_2 = h_i_2 * scale
        sum_log_scale += log_scale
    return h_i * np.exp(-x ** 2 / 2 + sum_log_scale)
```

I ditched my working with natural length and energy scales to match Bauke's code better. I guess this is done at the expense of losing some accuracy.
The H.O. eigenstates are now:

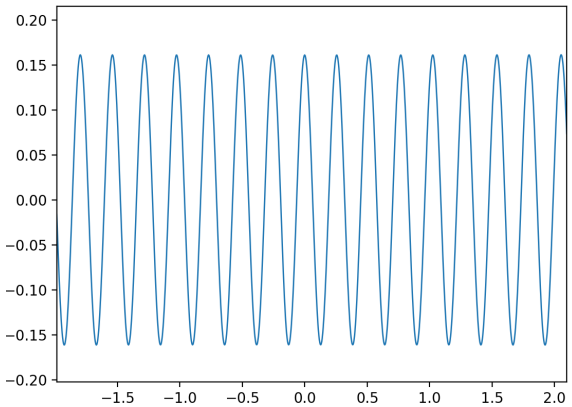
$$\psi_n(x) = \sqrt{\frac{2}{n}}\, x\, \psi_{n-1}(x) \sqrt{\frac{n-1}{n}} \psi_{n-2}(x)$$

I have a conditional indexed statement for the case when I use arrays' of x values (for example I would use a linspace object to plot the hermite functions.)
Since my underflow issues occur with x=0 then I set the x==0 to be instead a very small number.
I must check if I am actually obtaining some hermite functions for larger N=300 with the modified code from Bauke.
Previously I wasn't getting any results for N>100.
This is my function call as a plot

```
xs = np.linspace(-40, 40, 2048 * 10, endpoint=False)
plt.plot(xs, psi_blank(xs, 300))
plt.show()
```



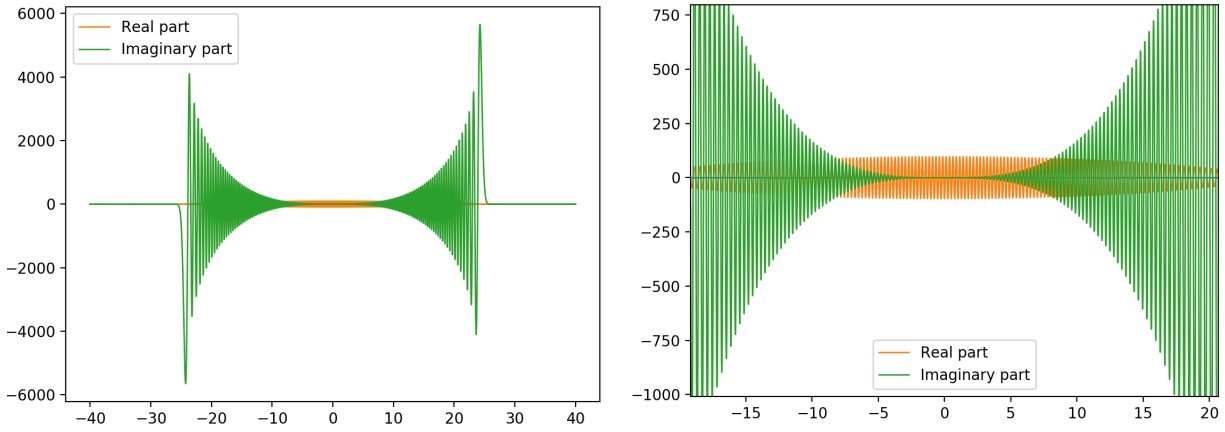
This is totally a harmonic oscillator eigenfunction for large N! YAY



and there are no discontinuities at x = 0 thanks to my fix.
I probably could reconsider the limits of integration to save time.

This is a plot of the complex Hamiltonian operating on the wavefunction

```
###################################function calls#################################################
# GLOBALS
ϵ = 1
k = 1 / 2
x = 2
N = 10

# ## NxN MATRIX
# Matrix = Matrix(x, N)
# np.save('matrix.npy', Matrix)
# print(f"\nMatrix\n{Matrix}")

xs = np.linspace(-40, 40, 2048 * 10, endpoint=False)
plt.plot(xs, np.real(Hamiltonian(xs, ϵ, 300)), label="Real part", linewidth=1)
plt.plot(xs, np.imag(Hamiltonian(xs, ϵ, 300)), label="Imaginary part", linewidth=1)
plt.legend()
plt.show()
```



Real part wiggles near the centre but decays as we move to the edges.

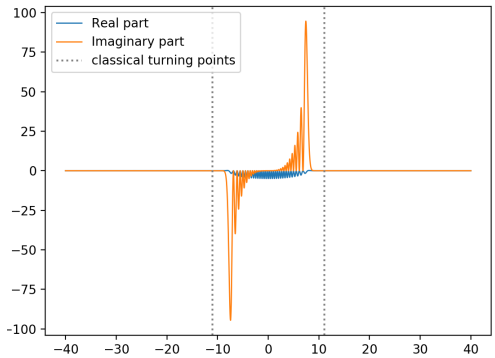
Optimization

These are plots of the integrand in

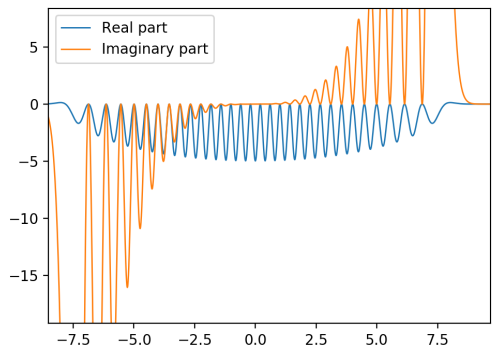
$$\langle n'|H|n\rangle = \int \psi^*_{n'}(x)\left(-\frac{d^2}{dx^2} + x^2(ix)^\epsilon\right)\psi_n(x)dx$$

I want to use these to figure out the most optimal choice for the integration limits
Given that currently I am using b = np.abs(np.sqrt(4 * max(m, n) + 2))
The integration limits change depending on the values of m and n.
I want to minimize the integration limits as much as possible to save time on the algorithm.

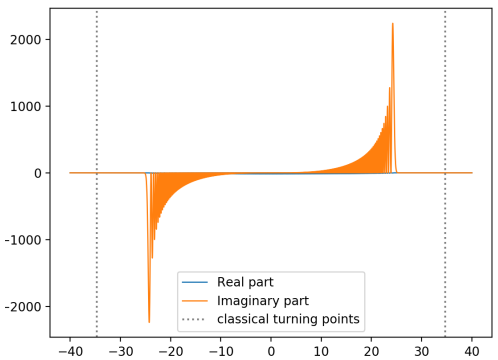
For m = 30 and n = 30



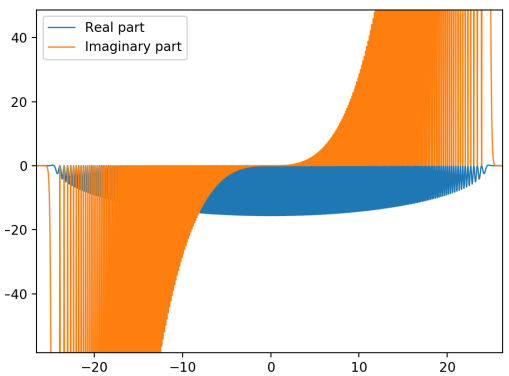
Function behaviour close-up



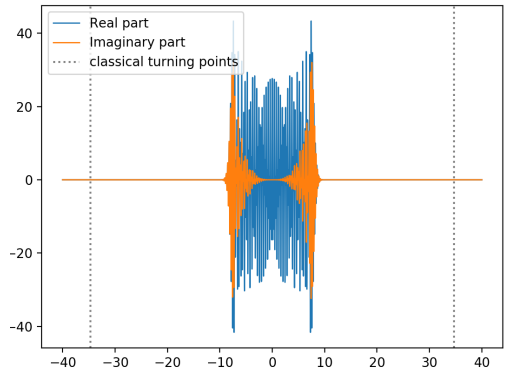
For m = 300 and n = 300



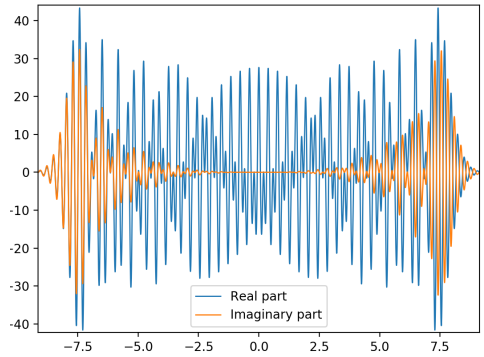
Function behaviour close-up



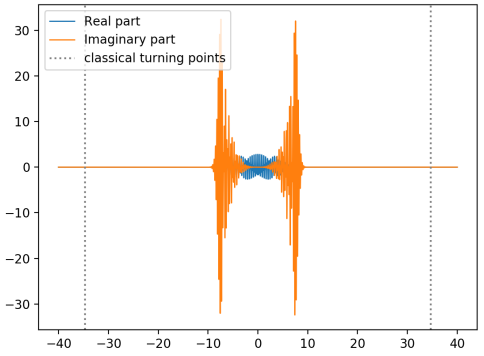
For m = 30 and n = 300



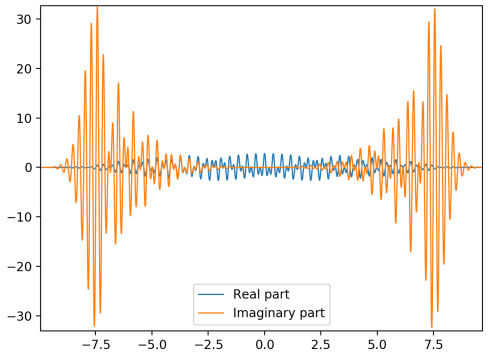
Function behaviour close-up



For m = 300 and n = 30



Function behaviour close-up



From these plots I can see that my integration limits are definitely larger than they need to be.
The difference between the turning point and the x-point after which my functions decay to ~1e-12 requires that I set my new “turning point” to be a function of the min(m,n) and in addition that I add two units more to this point.

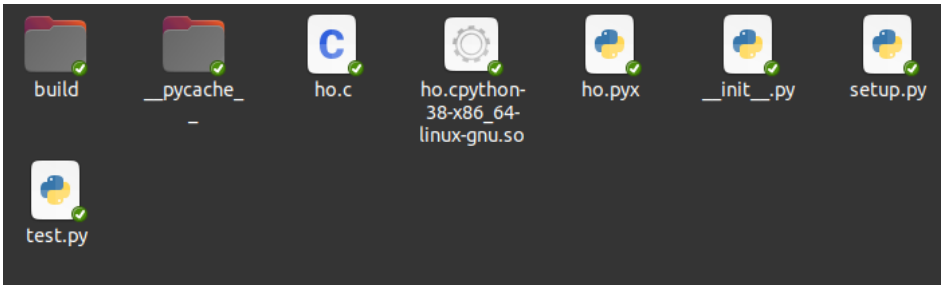
Ie. The new turning point is: b = np.abs(np.sqrt(4 * min(m, n) + 2)) + 2
Since the Hermite functions are up and running I’ve tried to calculate the matrix for the following parameters
GLOBALS
$\epsilon = 1$
k = 1 / 2
x = 2
N = 130

The issue I’ve encountered is that due to the complexity of the algorithm. The algorithm I wrote is N^3.
Therefore the time elapsed per run is huge.
Here is a picture of the elapsed time for each loop required in my algorithm (loops for m and n respectively)

Considering that I have to do the calculations for a large N and repeat several times for $\epsilon \in [-1, 0)$I am kinda screwed with the computational time I require to get my diagonalisable matrices.

Need to think about this for a while and see if there is a solution to this problem, otherwise I’ll have to try to solve the equation analytically. Even then I am unsure If I will not run into more problems.

09/05/21
I don’t think I can optimize the code by myself. I was chatting to Chris about this and he said that I would have to write some cython code in order to compile the function using C instead of using a more high level language loop. He very kindly offered to help me and wrote some cython code for me.



This is the function to be compiled in C:
ho.pyx

```
#cython: language_level=3
#cython: initializedcheck=False
#cython: boundscheck=False
#cython: wraparound=False
#cython: nonecheck=False
#cython: cdivision=True

import cython
from libc.math cimport sqrt, exp, pi, frexp, exp2, exp, log
cimport numpy as cnp
import numpy as np

cdef inline double _psi(int n, double x) nogil:
    cdef double h, h_prev, sum_log_scale, scale
    cdef int i, log_scale
    h_prev = pi ** -0.25
    h = sqrt(2.0) * x * pi ** -0.25
    sum_log_scale = 0
    if n == 0:
        h = h_prev
    for i in range(2, n + 1):
        h, h_prev = sqrt(2.0 / i) * x * h - sqrt(float(i - 1) / i) * h_prev, h
        frexp(h, &log_scale)
        scale = exp2(-log_scale)
        h *= scale
        h_prev *= scale
        sum_log_scale += log_scale
    return h * exp(-(x ** 2) / 2 + log(2) * sum_log_scale)


def psi(int n, x):
    cdef int i
    if isinstance(x, np.ndarray) and x.ndim:
        out = np.empty(x.shape)
        for i in range(x.size):
            out.flat[i] = _psi(n, x.flat[i])
        return out
    return _psi(n, x)
```

The algorithm is still the same complexity as before but the main speed up is due to the C compiling.
Chris even wrote a test.py file that he used to compare the speed of his compiled function vs the standard python function that I wrote.

```
[ana:odhobs]$ python3 test.py
testing correctness of first ten states
testing handling of dtypes
  np float
  np 0d float arr
  np int
  np 0d int arr
  pyint
  pyfloat
testing array speed with (1024,) array
  pytime: 0.05760979652404785
  cytime: 0.029085636138916016
  2.0× speedup
testing python loop speed with 1024 points
  pytime: 11.045326948165894
  cytime: 0.02923107147216797
  377.9× speedup
```

The code tests the correctness and speed of the calculation of HO basis states using an array of x values (size: (1024,)) and the basis states calculated for 1024 different x values. Basically a vectorised calculation and a single point calculation comparison for each of my python and his cython. His results absolutely destroy mine. His individual x calculations are ~378✕ faster than mine!

Let's see if I can do some calculations for some negative ϵ values now that the algorithm is faster!

```
k = 1 / 2
x = 2
N = 10

ϵ = -1
Matrix = Matrix(x, ϵ, N)
# np.save(f'matrix_{ϵ}.npy', Matrix)
print(f"\nMatrix\n{Matrix}")
```

This call throws an error about x = 0 raised to a negative power.

```
Traceback (most recent call last):
  File "Bending.py", line 89, in <module>
    Matrix = Matrix(x, ϵ, N)
  File "Bending.py", line 74, in Matrix
    element = complex_quad(
  File "Bending.py", line 22, in complex_quad
    real_integral = quad(real_func, a, b, **kwargs)
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 341, in quad
    retval = _quad(func, a, b, args, full_output, epsabs, epsrel, limit,
  File "/home/ana/.local/lib/python3.8/site-packages/scipy/integrate/quadpack.py", line 453, in _quad
    return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
  File "Bending.py", line 17, in real_func
    return np.real(func(*args))
  File "Bending.py", line 66, in element_integrand
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)
  File "Bending.py", line 57, in Hamiltonian
    print(f"{(x ** 2 * (1j * x) ** ϵ) = }")
ZeroDivisionError: 0.0 to a negative or complex power
```

It probably can be fixed like this

```
def Hamiltonian(x, ϵ, n):
    x = 1e-200
    h = 1e-6
    d2Ψdx2 = (cpsi_blank(n, x + h) - 2 * cpsi_blank(n, x) + cpsi_blank(n, x - h)) / h ** 2
    return d2Ψdx2 + (x ** 2 * (1j * x) ** ϵ) * cpsi_blank(n, x)
```

I will run this now

```
# GLOBALS
epsilons = np.linspace(-1, 0, 5)
k = 1 / 2
x = 2
N = 800

# NxN MATRIX
for ϵ in epsilons:
    print(f"{ϵ = }")
    matrix = Matrix(x, N)
    np.save(f'matrix_{ϵ}.npy', matrix)
    # print(f"\nMatrix\n{matrix}")
```

Important questions:

- How do I use Meera's suggestion for my Hamiltonian: $-\frac{d^2}{dx^2} + x^2 + x^2 + ((ix)^\epsilon - 1)x^2$?
- What does broken symmetry mean in this context?

Week 10 (10/05/21 – 16/05/21)

~~Aims:~~

1. ~~Reproduce Fig 1 in Bender~~
2. ~~Start preparing TALK (12 minutes)~~
    a. Due week 11 for tqm group presentation
    b. Due Week 12 Thursday 27/05 9:45am

Tasks:

1. ~~**PLOT THE BROKEN SYMMETRY SECTOR OF THE SPECTRUM IN FIG 1.**~~
2. Continue Gao's Hamiltonian analysis (using Benders methodology)

10/05/21
It has been around 10 hours since I started to evaluate matrix elements for $\epsilon = -1$.
I am planning to stop the loop as soon as this matrix finishes to be constructed and I want to test if the matrix has been saved correctly before I even continue with the other cases of ϵ.
I've noticed a strange pattern in the outermost loop evaluation.
The progress bars show that whenever the computer is calculating the matrix elements that have Hermite functions with even m values the progression of n values is slower than when m are odd.
I was able to screenshot 19 frames of the even m progress as opposed to only 3 frames for odd m (due to the calculation speed).

EVEN m

ODD m

I am thinking that I should plot the states and see what is going on in these specific instances.
The quad function might be faster when it encounters zeros and it probably extrapolates the results for most n values.

Finally the matrix for $\epsilon = -1.0$was calculated and I have verified that the eigenvalues of the matrix can be calculated.
To do this I've written the following code:

```
import numpy as np
from scipy import linalg
import matplotlib
import matplotlib.pyplot as plt

matrix_1 = np.load('matrix_-1.0.npy')

eigenvalues_1, eigenvectors_1 = linalg.eig(matrix_1)

non_zero_smol_imag = [i for i in eigenvalues_1 if abs(np.imag(i)) <= 1e-14 and np.real(i) !=0]
sorted_eigenvalues = sorted(non_zero_smol_imag, key=lambda x: x.real)

# print(f"\nEigenvalues ϵ: -1.0\n{eigenvalues_1}")
# print(f"\n{len(eigenvalues_1) = }\n")

# print(f"\n{len(non_zero_smol_imag) = }")
# print(f"\n{non_zero_smol_imag = }")

print("\nsorted_eigenvalues")
for i in sorted_eigenvalues:
    print(f"{i}")
```

The output is a sorted list of 46 elements with negligible imaginary parts.. I am not entirely sure that this is the final form of this block of code but it's a start, I'll see what needs to change when I have other matrices.

```
[ana:Bender]$ python3 testing_matrix.py

sorted_eigenvalues
(-11.56288052273166-4.03896783469992e-27j)
(-1.749851484823124-9.160379049172393e-25j)
(-0.2035533380837097-5.939347841309322e-20j)
(-0.017656207950259316+1.1547073766255144e-16j)
(-0.01073583156338766-1.3557436396998245e-16j)
(-0.008780260699709972-1.4163378551498074e-17j)
(-0.0004901281105750984-1.1243254919774825e-16j)
(-0.0001675112973132023 7-2.2956653351154463e-18j)
(-2.4379318214370532e-05+6.387178496784752e-17j)
(-1.8743007047188374e-05-8.662197967120669e-16j)
(-1.6160298300728615e-05+1.957811952484301e-15j)
(-1.2455995183768 99e-05+1.1851811771665759e-16j)
(-6.027058951822387e-06+7.566348207376808e-15j)
(-2.662444126723152e-06+8.466286888984437e-16j)
(-1.0713132922712913e-06+8.429182885244376e-15j)
(-2.980997528491373e-07-2.697527526611334e-15j)
(-3.1274463699112584e-08+1.394287131816683e-15j)
(-8.297206156985195e-11-6.1178007422633425e-19j)
(-2.576049960680563e-11+5.946401309484404e-19j)
(-1.114805150936068e-11+2.4238313362413216e-19j)
(-5.059122462334520 4e-12-2.0736708651767039e-19j)
(-2.089789381507422e-12-5.730295700960539e-21j)
(-1.2392968879558791e-12+2.55047595933408 93e-19j)
(-1.0935550552143188e-12-1.759843476371505e-19j)
(-1.7354335561412427e-13-7.4004068863357e-20j)
(1.3214317863662587e-12+8.584529611281396e-20j)
(3.863134625753219e-12+1.0488833396143189e-19j)
(1.238820692550907 2e-11+1.917324791673079e-19j)
(1.6501730249623268e-11+1.0291048455526441e-19j)
(1.8331756482804455e-11-5.754360726821477e-20j)
(6.79230740280569e-11+7.828613117201362e-19j)
(2.9357927348842027e-09-5.556647552376398e-17j)
(3.970173530580562e-08-2.5983155946106064e-15j)
(3.4139874060592524e-07+3.7584901025096285e-15j)
(1.3165012859056658e-06+1.2699406535518105e-15j)
(2.20673219859333 14e-06-2.975846175412076e-15j)
(4.103870785649867e-06+5.70012009814433e-15j)
(1.96792680102365e-05-9.344451642966653e-17j)
(2.3746233303162938e-05-3.598775896231381e-17j)
(0.0009314579256801611-5.60113312103318e-18j)
(0.0011661104189677484+2.318290032628509e-16j)
(0.003645799664372238-2.4685675407573667e-18j)
(0.005417428734682 15+3.880614723075615e-17j)
(0.010233891648908044-8.673617379884035e-18j)
(0.02162069396780208-9.173203764921726e-16j)
(4.69746534704189+8.158715026166573e-26j)
[ana:Bender]$
```

11/05/21

Bug fixes

I discovered that my last fix to raising x == 0 to a negative power by setting x == 1e-200 was actually not the right fix.
Since I am using quad to integrate over several values of x between the modified turning points of the classical harmonic oscillator, setting the strict single value of x within the Hamiltonian function I use in my integrand made my states to exclusively use x == 1e-200 for every value of x.
I discovered this by comparing the sorted eigenvalues of matrices $\epsilon = -1.0$ and $\epsilon = -0.75$and discovering that they were identical!
Tragic because that means that I've wasted 2 days for a calculation of the same thing over and over.

```
sorted_eigenvalues for matrix_ϵ = -1.0

len(sorted_eigenvalues) = 46
(-11.56288052273166-4.03896783469992e-27j)
(-1.749851484823124-9.160379049172393e-25j)
(-0.2035533380837097-5.939347841309322e-20j)
(-0.017656207950259316+1.1547073766255144e-16j)
(-0.01073583156338766-1.3557436396998245e-16j)
(-0.008780260699709972-1.4163378551498074e-17j)
(-0.0004901281105750984-1.1243254919774825e-16j)
(-0.00016751129731320237-2.2956653351154463e-18j)
(-2.4379318214370532e-05+6.387178496784752e-17j)
(-1.8743007047188374e-05-8.662197967120669e-16j)
(-1.6160298300728615e-05+1.957811952484301e-15j)
(-1.2455995183768990e-05+1.1851811771665759e-16j)
(-6.027058951822387e-06+7.566348207376808e-15j)
(-2.662444126723152e-06+8.466286888984437e-16j)
(-1.0713132922712913e-06+8.429182885244376e-15j)
(-2.980997528491373e-07-2.697527526611334e-15j)
(-3.1274463699112584e-08+1.394287131816683e-15j)
(-8.297206156985195e-11-6.1178007422633425e-19j)
(-2.576049960680563e-11+5.946401309484404e-19j)
(-1.114805150936068e-11+2.4238313362413216e-19j)
(-5.0591224623345204e-12-2.07367086517670390e-19j)
(-2.089789381507422e-12-5.730295700960539e-21j)
(-1.2392968879558791e-12+2.5504759593340893e-19j)
(-1.0935550552143188e-12-1.759843476371505e-19j)
(-1.7354335561412427e-13-7.40040688633570e-20j)
(1.3214317863662587e-12+8.584529611281396e-20j)
(3.863134625753219e-12+1.0488833396143189e-19j)
(1.2388206925509072e-11+1.917324791673079e-19j)
(1.6501730249623268e-11+1.0291048455526441e-19j)
(1.8331756482804455e-11-5.754360726821477e-20j)
(6.79230740280569e-11+7.828613117201362e-19j)
(2.9357927348842027e-09-5.556647552376398e-17j)
(3.970173530580562e-08-2.5983155946106064e-15j)
(3.4139874060592524e-07+3.7584901025096285e-15j)
(1.3165012859056658e-06+1.2699406535518105e-15j)
(2.2067321985933314e-06-2.975846175412076e-15j)
(4.103870785649867e-06+5.70012009814433e-15j)
(1.96792680102365e-05-9.344451642966653e-17j)
(2.3746233303162938e-05-3.598775896231381e-17j)
(0.0009314579256801611-5.60113312103318e-18j)
(0.0011661104189677484+2.318290032628509e-16j)
(0.003645799664372238-2.4685675407573667e-18j)
(0.005417428734681250+3.880614723075615e-17j)
(0.010233891648908044-8.673617379884035e-18j)
(0.021620693967802080-9.173203764921726e-16j)
(4.697465347041890+8.158715026166573e-26j)
```

```
sorted_eigenvalues for matrix_(ϵ = -0.75)

len(sorted_eigenvalues) = 46
(-11.56288052273166-4.03896783469992e-27j)
(-1.749851484823124-9.160379049172393e-25j)
(-0.2035533380837097-5.939347841309322e-20j)
(-0.017656207950259316+1.1547073766255144e-16j)
(-0.01073583156338766-1.3557436396998245e-16j)
(-0.008780260699709972-1.4163378551498074e-17j)
(-0.0004901281105750984-1.1243254919774825e-16j)
(-0.00016751129731320237-2.2956653351154463e-18j)
(-2.4379318214370532e-05+6.387178496784752e-17j)
(-1.8743007047188374e-05-8.662197967120669e-16j)
(-1.6160298300728615e-05+1.957811952484301e-15j)
(-1.2455995183768990e-05+1.1851811771665759e-16j)
(-6.027058951822387e-06+7.566348207376808e-15j)
(-2.662444126723152e-06+8.466286888984437e-16j)
(-1.0713132922712913e-06+8.429182885244376e-15j)
(-2.980997528491373e-07-2.697527526611334e-15j)
(-3.1274463699112584e-08+1.394287131816683e-15j)
(-8.297206156985195e-11-6.1178007422633425e-19j)
(-2.576049960680563e-11+5.946401309484404e-19j)
(-1.114805150936068e-11+2.4238313362413216e-19j)
(-5.0591224623345204e-12-2.07367086517670390e-19j)
(-2.089789381507422e-12-5.730295700960539e-21j)
(-1.2392968879558791e-12+2.5504759593340893e-19j)
(-1.0935550552143188e-12-1.759843476371505e-19j)
(-1.7354335561412427e-13-7.40040688633570e-20j)
(1.3214317863662587e-12+8.584529611281396e-20j)
(3.863134625753219e-12+1.0488833396143189e-19j)
(1.2388206925509072e-11+1.917324791673079e-19j)
(1.6501730249623268e-11+1.0291048455526441e-19j)
(1.8331756482804455e-11-5.754360726821477e-20j)
(6.79230740280569e-11+7.828613117201362e-19j)
(2.9357927348842027e-09-5.556647552376398e-17j)
(3.970173530580562e-08-2.5983155946106064e-15j)
(3.4139874060592524e-07+3.7584901025096285e-15j)
(1.3165012859056658e-06+1.2699406535518105e-15j)
(2.2067321985933314e-06-2.975846175412076e-15j)
(4.103870785649867e-06+5.70012009814433e-15j)
(1.96792680102365e-05-9.344451642966653e-17j)
(2.3746233303162938e-05-3.598775896231381e-17j)
(0.0009314579256801611-5.60113312103318e-18j)
(0.0011661104189677484+2.318290032628509e-16j)
(0.003645799664372238-2.4685675407573667e-18j)
(0.005417428734681250+3.880614723075615e-17j)
(0.010233891648908044-8.673617379884035e-18j)
(0.021620693967802080-9.173203764921726e-16j)
(4.697465347041890+8.158715026166573e-26j)
```

Anyway, at least I discovered the issue and fixed it.
Another issue I fixed in this block was assigning a to a variable named psi_n the calculated eigenstate cpsi_blank(n, x)
This should reduce the computation of the Hamiltonian function by a factor of 0.25 since within this function there were 4 function calls and I now have only 3.
The newest Hamiltonian function in my code takes an x as input and it locally redefines it as an np.array object which can then be indexed with an array of Booleans for the following condition: If some of the values in the array are zeroes the indexing notation reassigns the value 1e-200 to those values to avoid any divisions by zero.

```python
def Hamiltonian(x, ϵ, n):
    x = np.array(x)
    x[x == 0] = 1e-200
    h = 1e-6
    psi_n = cpsi_blank(n, x)
    d2Ψdx2 = (cpsi_blank(n, x + h) - 2 * psi_n + cpsi_blank(n, x - h)) / h ** 2
    return d2Ψdx2 + (x ** 2 * (1j * x) ** ϵ) * psi_n
```

I decided to run my code with a smaller N value (N = 100) to verify that everything is running correctly with minimal computational time waste.
(something I should've done from the start…)

Another observed side effect from this new fix is that now there is no discrepancy between the calculation speed between the odd and even m values.

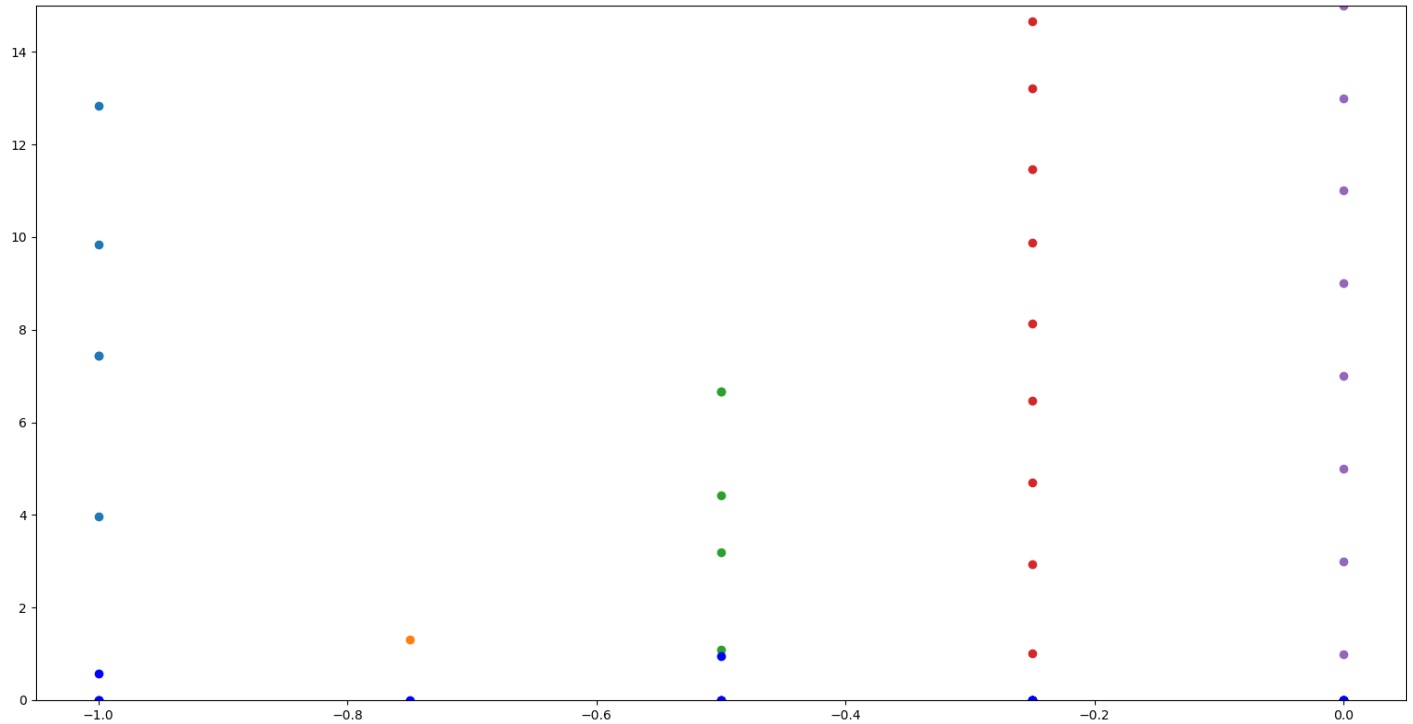
```
[ana:Bender]$ python3 -i Bending.py
ϵ = -1.0
  0%|                                                  | 0/100 [00:00<?, ?it/sBending.py:22: IntegrationWarning: The occurrence of roundoff error is detected, which prevents
  the requested tolerance from being achieved.  The error may be
  underestimated.
  real_integral = quad(real_func, a, b, **kwargs)
  7%|▋                                                 | 7/100 [00:21<04:47,  3.10s/itBending.py:22: IntegrationWarning: The integral is probably divergent, or slowly convergent.
  real_integral = quad(real_func, a, b, **kwargs)
100%|██████████████████████████████████████████████████| 100/100 [08:48<00:00,  6.84s/it]
ϵ = -0.75
100%|██████████████████████████████████████████████████| 100/100 [00:07<00:00, 10.20it/s]
100%|██████████████████████████████████████████████████| 100/100 [09:12<00:00,  7.11s/it]
ϵ = -0.5
100%|██████████████████████████████████████████████████| 100/100 [00:07<00:00,  9.94it/s]
100%|██████████████████████████████████████████████████| 100/100 [09:54<00:00,  7.51s/it]
ϵ = -0.25
100%|██████████████████████████████████████████████████| 100/100 [00:08<00:00,  9.28it/s]
100%|██████████████████████████████████████████████████| 100/100 [09:43<00:00,  7.41s/it]
ϵ = 0.0
100%|██████████████████████████████████████████████████| 100/100 [00:07<00:00, 13.50it/s]
100%|██████████████████████████████████████████████████| 100/100 [06:14<00:00,  4.27s/it]
>>>
```

Ugh, my Hamiltonian function was missing a negative sign on the second derivative...

```
return -d2Ψdx2 + (x ** 2 * (1j * x) ** ϵ) * psi_n
```

I tried plotting some sorted eigenvalues and they didn't look quite right.

I've fixed the issue and this is the result for 5 values of epsilon in the desired interval



The dark blue dots are the imaginary parts of each of the complex eigenvalues in the interval, the other colours represent the real parts (unfortunately Python chooses lighter blue to represent the real parts of the eigenvalues corresponding to epsilon -1… but the blue value is a bit different.).
This plot looks very promising. It is super similar to FIG 3. in the following paper:
r by CM Bender et al: http://dx.doi.org/10.1103/PhysRevA.95.052113

Behavior of eigenvalues in a region of broken-PT symmetry

Abstract (Bender et al. 1)

> $\mathcal{PT}$-symmetric quantum mechanics began with a study of the Hamiltonian $H = p^2 + x^2(ix)^\varepsilon$. When $\varepsilon \geq 0$, the eigenvalues of this non-Hermitian Hamiltonian are discrete, real, and positive. This portion of parameter space is known as the region of *unbroken* $\mathcal{PT}$ symmetry. In the region of *broken* $\mathcal{PT}$ symmetry $\varepsilon < 0$ only a finite number of eigenvalues are real and the remaining eigenvalues appear as complex-conjugate pairs. The region of unbroken $\mathcal{PT}$ symmetry has been studied but the region of broken $\mathcal{PT}$ symmetry has thus far been unexplored. This paper presents a detailed numerical and analytical examination of the behavior of the eigenvalues for $-4 < \varepsilon < 0$. In particular, it reports the discovery of an infinite-order exceptional point at $\varepsilon = -1$, a transition from a discrete spectrum to a partially continuous spectrum at $\varepsilon = -2$, a transition at the Coulomb value $\varepsilon = -3$, and the behavior of the eigenvalues as ε approaches the conformal limit $\varepsilon = -4$.

Notes from (Bender et al.):

A quantum-mechanical potential of form $\hat{x}^2\,(i\,\hat{x})^\epsilon$ does not necessarily lead to complex eigenvalues because the quantity *ix* is PT invariant. Indeed, the non-Hermitian PT -symmetric Hamiltonian.

$$H = \hat{p}^2 + \hat{x}^2\,(i\,\hat{x})^\epsilon$$

has the property that its eigenvalues are entirely real, positive, and discrete when ε ≥ 0.

A particularly interesting feature of PT -symmetric Hamiltonians is that they often exhibit a transition from a parametric region of unbroken PT symmetry in which all of the eigenvalues are real to a region of broken PT - symmetry in which some of the eigenvalues are real and the rest of the eigenvalues occur in complex-conjugate pairs. This transition occurs in both the classical and the quantized versions of a PT -symmetric Hamiltonian, and has been observed in numerous laboratory experiments (Bender et al. 1)

Exceptional points

Essentially nothing has been published regarding the analytic behavior of the complex eigenvalues as functions of ε in the region of broken PT- symmetry. However, it is known that there is a sequence of negative real values of ε lying between −1 and 0 at which pairs of real eigenvalues become degenerate and split into pairs of complex-conjugate eigenvalues. These special values of $\epsilon$are known as **exceptional points** (Bender et al. 2).
In general, eigenvalues usually have square-root branch-point singularities at exceptional points.

The distinct eigenvalues of a Hamiltonian may correspond with the sheets of a Riemann surface. Interestingly, it is possible to vary the parameters of a Hamiltonian in laboratory experiments and thus to observe experimentally the effect of encircling exceptional points.

The figure from Bender's that I've been working displays an analytic continuation for the broken symmetry region ($\epsilon < 0$) to the real eigenvalues in the unbroken symmetry region ($\epsilon \geq 0$). Bender's Figure 1. Also displays an ***infinite-order exceptional point*** at $\epsilon = -1$(Bender et al. 2).

Infinite-order exceptional point

At $\epsilon = -1$there is an elaborate **logarithmic** spiral (double-helix) of eigenvalues:

What happens in this logarithmic singularity?
The real part of each complex conjugate pair of eigenvalues that is formed at exceptional points between $\epsilon = -1$and $\epsilon = 0$ approaches +∞ like $|\ln(\varepsilon + 1)|^{2/3}$as ε approaches −1.
In contrast, **the imaginary parts of each pair of eigenvalues vanish logarithmically at ε = −1.**
As ε goes below −1, the real parts
of the eigenvalues once again become finite and the imaginary parts of the eigenvalues rise up from 0. As ε goes from just above to just below −1, the imaginary parts of the eigenvalues appear to undergo discrete jumps but in fact they vary continuously as functions of (Bender et al. 2).

Eigenvalue Behaviour as $\epsilon \to -1$ (Bender et al. 3).

The TISE for the PT-symmetric non-Hermitian Hamiltonians I am studying are characterised by boundary conditions imposed on the eigenfunctions requiring that $\psi(x) \to 0$ exponentially rapidly as |x| → ∞ in a pair of **Stokes wedges** in the complex-x plane. This subsection explains the locations of these Stokes wedges.
It is necessary to introduce a branch cut chosen to run from 0 to ∞ in the complex-x plane along the positive imaginary axis because this choice respects the PT symmetry of the Hamiltonian. This is because PT symmetry translates into left-right symmetry in the complex-x plane (mirror symmetry with respect to the imaginary-x axis)

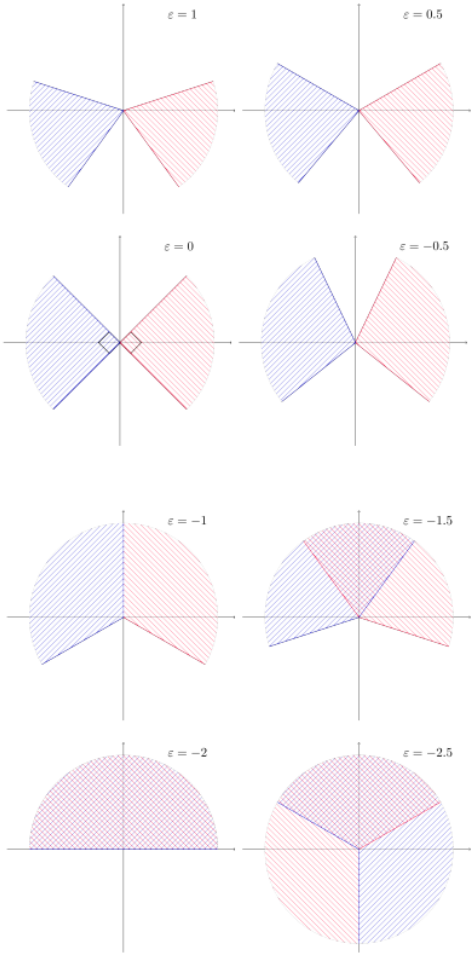


FIG. 2: [Color online] Stokes wedges associated with the eigenvalue problem for the Hamiltonian $H = p^2 + x^2(ix)^\varepsilon$ for eight values of ε. The locations of center lines, the upper edges, and the lower edges of the Stokes wedges are given in (3)-(5). The left wedge is colored blue and the right wedge is colored red. As ε decreases, the wedges get wider and rotate upwards. At $\varepsilon = -1$ the two wedges touch and fuse into one wedge. However, when ε goes below -1, the sheets are again distinct; the left wedge rotates clockwise into sheet -1 and the right wedge rotates anticlockwise into sheet 1.

At the special value ε = −1 the logarithmic Riemann surface collapses to a single sheet; the wedges fuse and are no longer separated. As a result there are no eigenvalues at all (the spectrum is null)

Numerical behaviour of eigenvalues as ϵdecreases below 0 (Bender et al. 4).

In Figure 1. In Bender it may seem that the real eigenvalues disappear pairwise at special isolated values of ε. BUT in fact the eigenvalues do not disappear; rather, each pair of real eigenvalues fuse. These eigenvalues convert into a complex-conjugate pair of eigenvalues.

At this transformation point both the real and the imaginary parts of each pair of eigenvalues vary continuously; the real parts remain nonzero and the imaginary parts move away from zero as ε goes below the transition point.

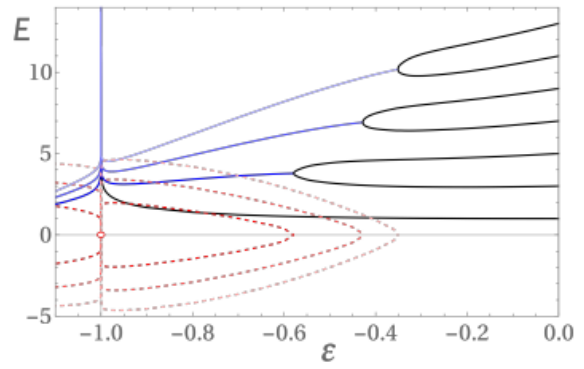


FIG. 3: [Color online] Eigenvalues of the Hamiltonian $H = p^2 + x^2(ix)^\varepsilon$ plotted as functions of the parameter ε for $-1.1 < \varepsilon < 0$. This graph is a continuation of the graph in Fig. 1. As ε decreases below 0 and enters the region of broken $\mathcal{PT}$ symmetry, real eigenvalues (solid black lines) become degenerate and then form complex-conjugate pairs. The real parts of these pairs of eigenvalues (solid blue lines) initially decrease as ε decreases but blow up suddenly as ε approaches -1. The real parts then decrease as ε decreases below -1. The imaginary parts of the eigenvalue pairs (dashed red lines) remain finite and appear to suffer discontinuous jumps at $\varepsilon = -1$. However, a closer look shows that these dashed lines rapidly decay to 0 near $\varepsilon = -1$ and then rapidly come back up to different values as ε passes through -1. A blow-up of the region near $\varepsilon = -1$ is given in Figs. 4.

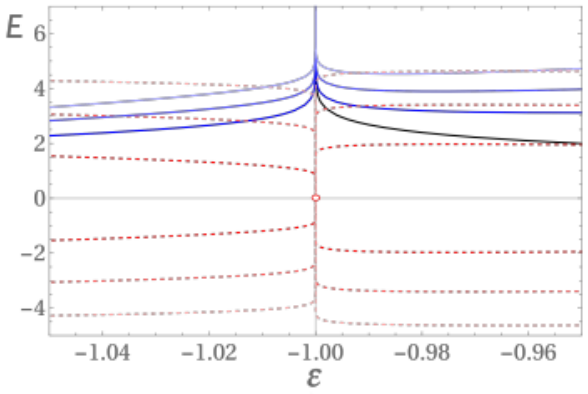


FIG. 4: [Color online] Detailed view of Fig. 3 showing the behavior of the eigenvalues of the Hamiltonian $H = p^2 + x^2(ix)^\varepsilon$ plotted as functions of the parameter ε for $-1.05 \leq \varepsilon \leq -0.95$. There is one real eigenvalue for $\varepsilon > -1$ (solid black line). The real parts of the complex eigenvalues (blue solid lines) and the real eigenvalue diverge at $\varepsilon = -1$. The complex eigenvalues occur in complex-conjugate pairs and the imaginary parts of the eigenvalues rapidly go to 0 at $\varepsilon = -1$. These behaviors are expressed quantitatively in (14).

The calculation of my $N \times N$matrices $(N = 100)$ took me around 40 minutes so I decided to calculate 100 matrices for different epsilon values in the interval … hopefully these calculations will be done by tomorrow morning.
This is the code I'll run tonight.

```python
def Hamiltonian(x, ϵ, n):
    x = np.array(x)
    x[x == 0] = 1e-200
    h = 1e-6
    psi_n = cpsi_blank(n, x)
    d2Ψdx2 = (cpsi_blank(n, x + h) - 2 * psi_n + cpsi_blank(n, x - h)) / h ** 2
    return -d2Ψdx2 + (x ** 2 * (1j * x) ** ϵ) * psi_n


def element_integrand(x, ϵ, m, n):
    # CHECK THESE IF mass = 1 instead of 1/2
    psi_m = cpsi_blank(m, x)
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)


# NxN MATRIX
def Matrix(x, N):
    M = np.zeros((N, N), dtype="complex")
    for m in tqdm(range(N)):
        for n in tqdm(range(N)):
            b = np.abs(np.sqrt(4 * min(m, n) + 2)) + 2
            element = complex_quad(
                element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-08, limit=1000
            )
            # print(element)
            M[m][n] = element
    return M


#################################function calls#############################################
# GLOBALS
epsilons = np.linspace(-1, 0, 100)
k = 1 / 2
x = 2
N = 100

# NxN MATRIX
for i, ϵ in enumerate(epsilons):
    print(f"{ϵ = }")
    matrix = Matrix(x, N)
    np.save(f'matrices/matrix_{i:03d}.npy', matrix)
```

WHAT AM I ACTUALLY DOING?

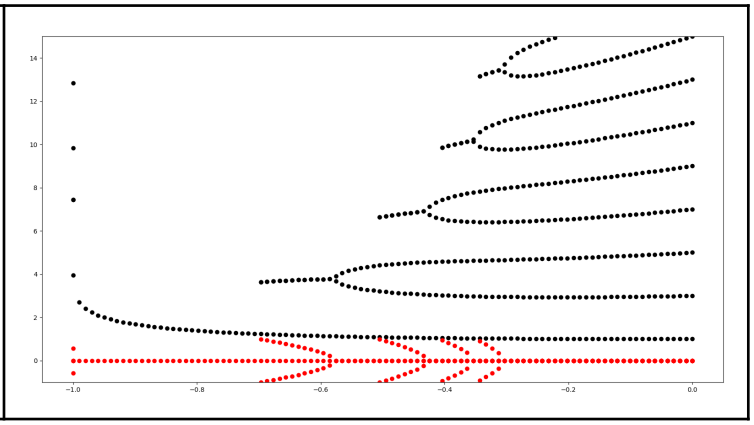
1. Need to find the general form of the H.O. basis functions ✅
   a. Write these in the coordinate basis ✅
2. Decide how I am solving the equation ✅
   Numerically
3. How many functions am I using? ✅
   N=100
4. Make the Hamiltonian matrices (100 matrices) ✅
5. Diagonalise the Hamiltonian matrices ✅
6. Plot eigenvalues as functions of ϵ ✅
   Plot real and imaginary parts to simulate fig 3. above

12/05/21
MEETING DAY!
My code finally finished running, these are the time stats:

```
real    917m52.861s
user    917m18.188s
sys     0m14.315s
[ana:Bender]$
```

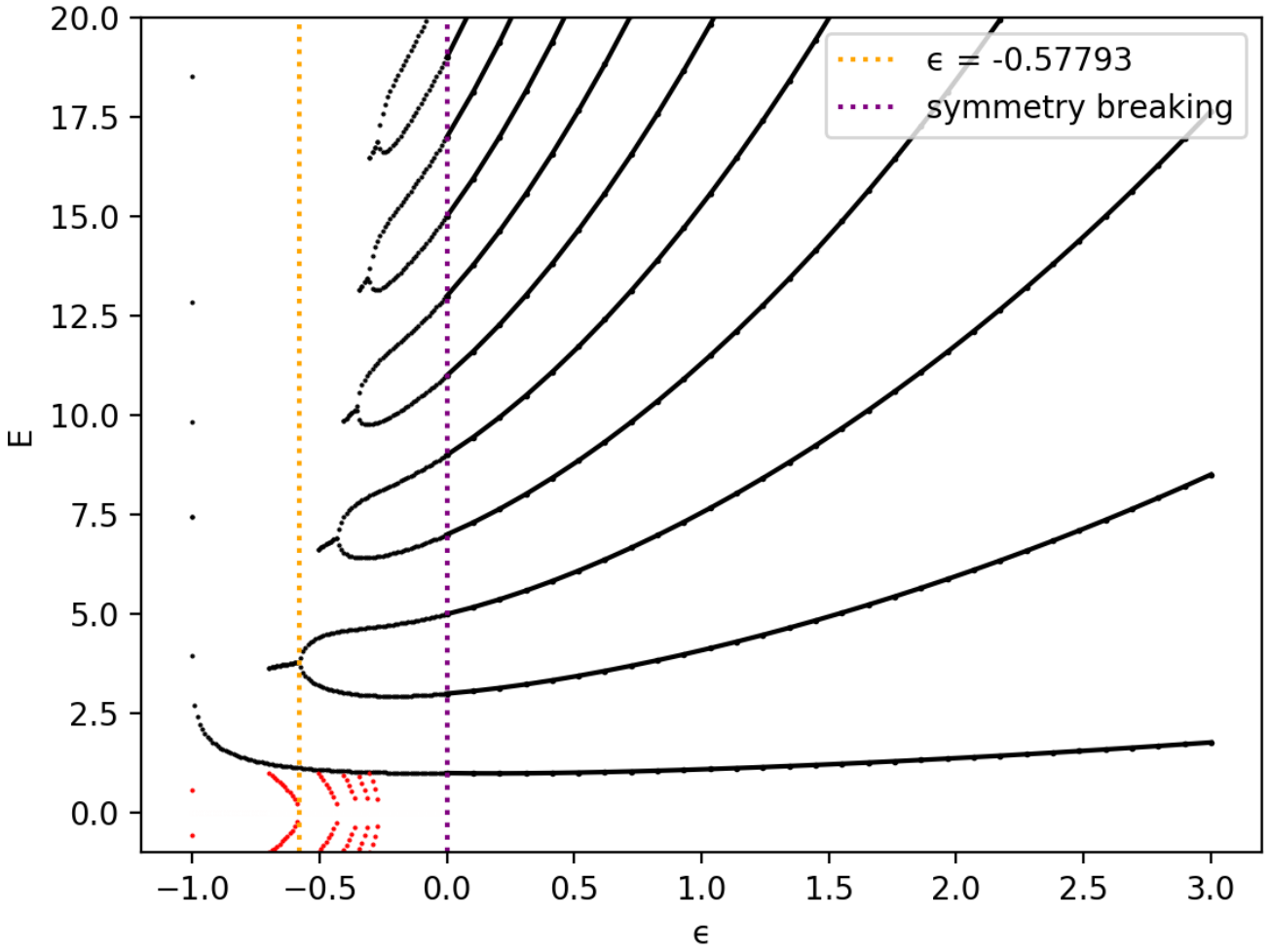
This is the broken symmetry region!

```python
import numpy as np
from scipy import linalg
import matplotlib
import matplotlib.pyplot as plt

N = 100
epsilons = np.linspace(-1.0, 0, N)

for i, ϵ in enumerate(epsilons):
    matrix = np.load(f'matrices/matrix_{i:03d}.npy')
    eigenvalues, blergh_eigenvectors = linalg.eig(matrix)

    positive_evals = [i for i in eigenvalues if 0 < np.real(i) < 15 and abs(np.imag(i)) < 1]
    sorted_eigenvalues = sorted(positive_evals, key=lambda x: np.real(x))

    ϵ_list = np.full(len(sorted_eigenvalues), ϵ)
    # print(f"{ϵ_list = }")

    plt.plot(ϵ_list, np.real(sorted_eigenvalues), marker='o', linestyle='None', color='k', markersize=6)
    plt.plot(ϵ_list, np.imag(sorted_eigenvalues), marker='o', linestyle='None', color='r', markersize=6)

plt.axis(ymin=-1, ymax=15)
plt.axis()
plt.show()
```

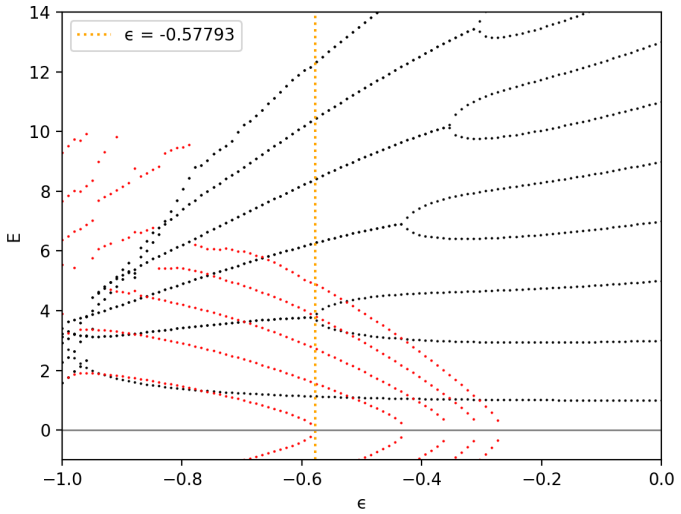


```python
######################## FINAL plot of eigenvalues ###################################
# ITERATIVE approach 2
Energies_2 = []
for n in range(10):
    E_s = []
    for ϵ in np.linspace(0, 3, 30):
        E = complex_fsolve(error, E1, args=(ϵ, n))
        E_s.append(E)
        # print(f" {ϵ = }, {n = }, {E = }")
    Energies_2.append(E_s)
# print(Energies_2)

#PLOTING ITERATIVE approach 1
for E_ϵs in Energies_2:
    # print(E_ϵs)
    ϵ = np.linspace(0, 3, 30)
    plt.plot(ϵ, E_ϵs, "o-", color='k', markersize=1)
# plt.legend()
# plt.ylim(0, 20)
plt.xlabel("ϵ")
plt.ylabel("E")

for i, ϵ in enumerate(epsilons):
    matrix = np.load(f'matrices/matrix_{i:03d}.npy')
    eigenvalues, blergh_eigenvectors = linalg.eig(matrix)

    positive_evals = [i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) < 1]
    sorted_eigenvalues = sorted(positive_evals, key=lambda x: np.real(x))

    ϵ_list = np.full(len(sorted_eigenvalues), ϵ)
    zeroes_list = np.zeros_like(ϵ_list)

    plt.plot(ϵ_list, np.real(sorted_eigenvalues), marker='.', linestyle='None', color='k', markersize=1)
    plt.plot(ϵ_list, np.imag(sorted_eigenvalues), marker='.', linestyle='None', color='r', markersize=1)
    plt.plot(ϵ_list, zeroes_list, marker='.', linestyle='None', color='white', markersize=2)

plt.axis(ymin=-1, ymax=20)
plt.axvline(-0.57793, color='orange', linestyle=':', label="ϵ = -0.57793")
plt.axvline(0, color='purple', linestyle=':', label="PT-symmetry breaking")
plt.legend()
plt.show()
```

My rendition of Bender's figure 1.

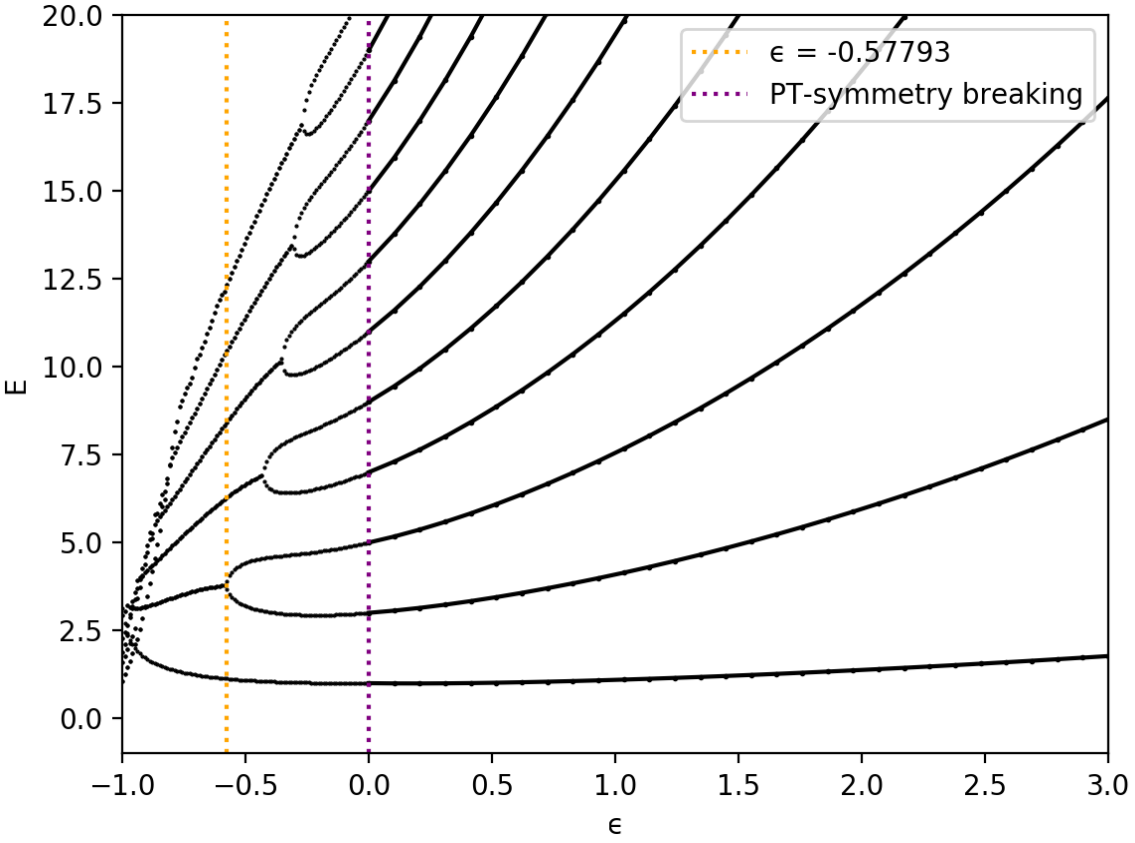



```python
Energies_2 = np.load("Energies_unbroken.npy")

def final_form():
    for E_es in Energies_2:
        # print(E_es)
        ϵ = np.linspace(0, 3, 30)
        # plt.plot(ϵ, E_es, "o-", color='k', markersize=1)

    eigenvectors_list = []
    for i, ϵ in enumerate(epsilons):
        matrix = np.load(f'matrices/matrix_{i:03d}.npy')
        eigenvalues, eigenvectors = linalg.eig(matrix)
        eigenvectors_list.append(eigenvectors)

        positive_evals = [
            i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) < 10
        ]
        sorted_eigenvalues = sorted(positive_evals, key=lambda x: np.real(x))
        sorted_eigenvalues = sorted_eigenvalues[:11]

        ϵ_list = np.full(len(sorted_eigenvalues), ϵ)

        plt.plot(
            ϵ_list,
            np.real(sorted_eigenvalues),
            marker='.',
            linestyle='None',
            color='k',
            markersize=1,
        )
        mask_imag = 1e-6 < abs(np.imag(sorted_eigenvalues))
        plt.plot(
            ϵ_list[mask_imag],
            np.imag(sorted_eigenvalues)[mask_imag],
            marker='.',
            linestyle='None',
            color='r',
            markersize=1,
        )

    plt.axis(xmin=-1, xmax=0, ymin=-1, ymax=14)
    plt.axvline(-0.57793, color='orange', linestyle=':', label="ϵ = -0.57793")
    plt.axhline(0, color='grey', linestyle='-', linewidth=1)
    # plt.axvline(0, color='purple', linestyle=':', label="PT-symmetry breaking")
    plt.legend()
    plt.xlabel("ϵ")
    plt.ylabel("E")
    plt.savefig("NHH_eigenvalues.png")
    plt.show()
    return eigenvectors_list

coefficients = final_form()
```



My numerics appears to break as $\epsilon \to -1$

If I plot only the real parts (ie. without any selection criteria for the imaginary parts)

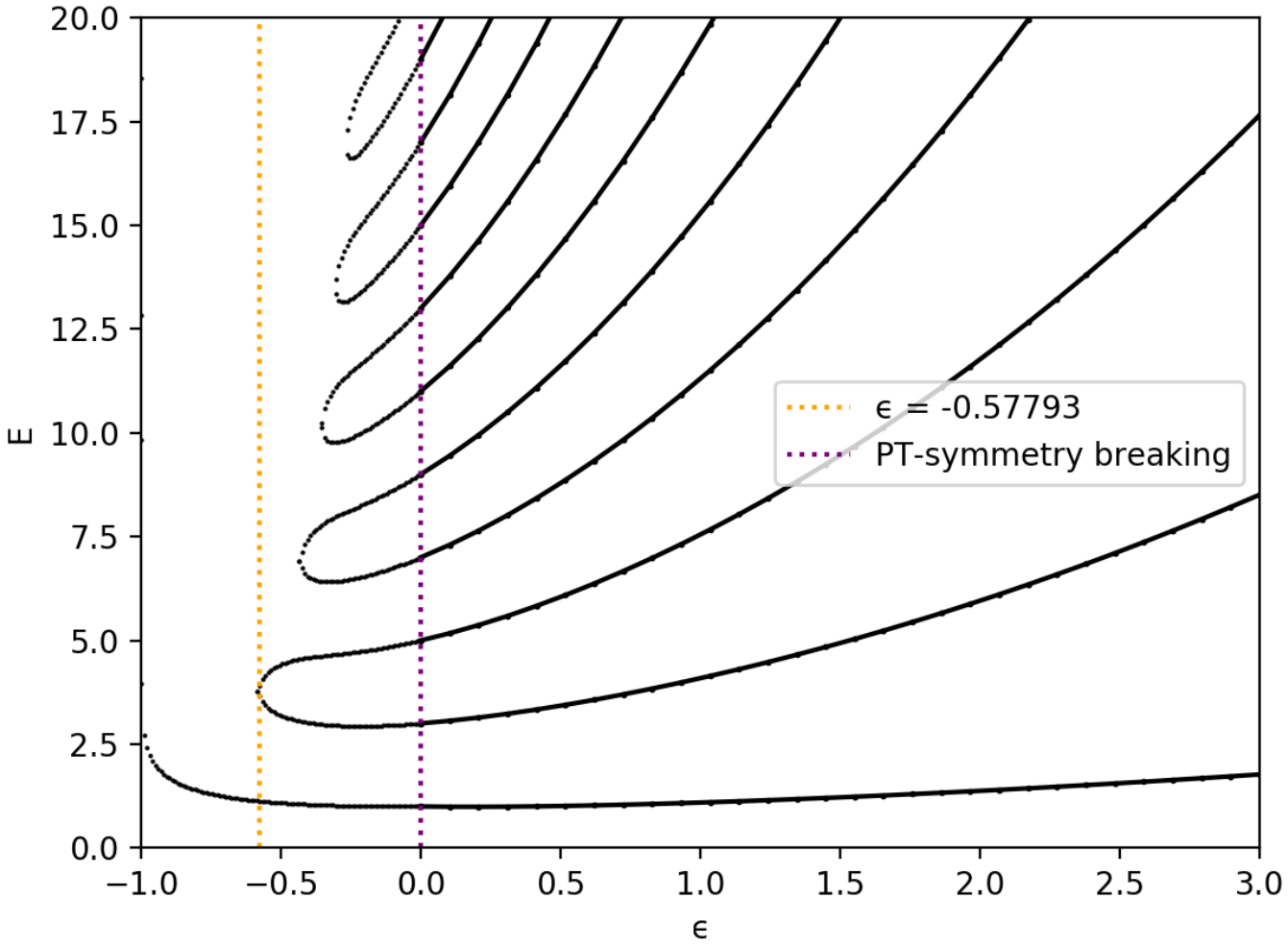


As I stated before numerics seem to fail the closer $\epsilon \to -1$.

Figure 1. Final form

After trial and error The final condition imposed on the indexing is

```
positive_evals = [i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) < 0.3]
```



13/05/21

Eigenvectors

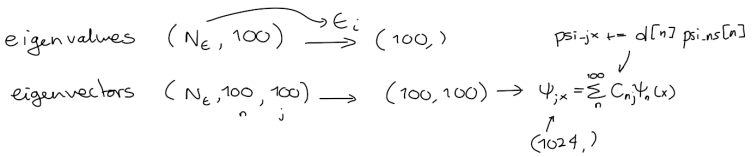
The question in everyone's minds is are the eigenvectors of these matrices orthogonal?

15/05/21

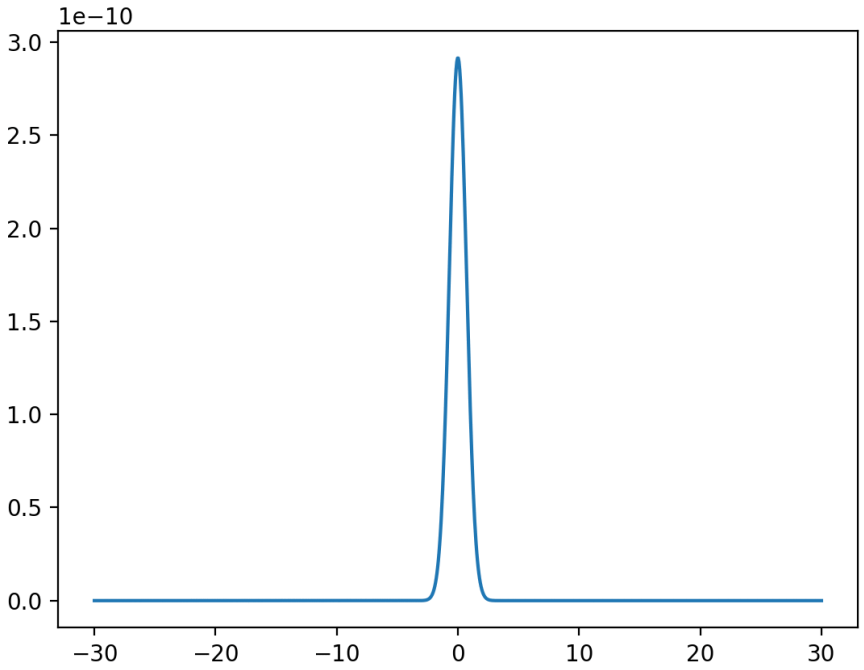
After a lot of thought, much confusion and many bugs… I wrote this working function.

```
coefficients = figure_final_form()

###################### Eigenvectors plot ################################

def spatial_wavefunctions(N, x, epsilons):
    x[x == 0] = 1e-200
    PSI_ns = []
    for n in range(N):
        psi_n = cpsi_blank(n, x)
        PSI_ns.append(psi_n)
    PSI_ns = np.array(PSI_ns)
    np.save(f"PSI_ns.npy", PSI_ns)

    eigenstates = []
    for i, ϵ in enumerate(epsilons):
        c = coefficients[i]
        for j in range(N):  # for each eigenvector
            d = c[:, j]
            psi_jx = np.zeros(x.shape, complex)
            for n in range(N):  # for each H.O. basis vector
                psi_jx += d[n] * PSI_ns[n]
                plt.plot(x, abs(psi_jx) ** 2)
                plt.savefig(f"spatial_wavefunctions/wavefunction_{ϵ:03d}_{n:03d}.png")
                plt.clf()
            assert 0
            eigenstates.append(psi_jx)

    np.save(f'eigenstates.npy', np.array(eigenstates))

N = 100
epsilons = np.linspace(-1.0, 0, N)
xs = np.linspace(-30, 30, 1024)
spatial_wavefunctions(N, xs, epsilons)

###################### Eigenvectors plot ################################
```

Here I calculate and plot the eigenstates for each epsilon value in my 100 ϵ between -1 and 0.

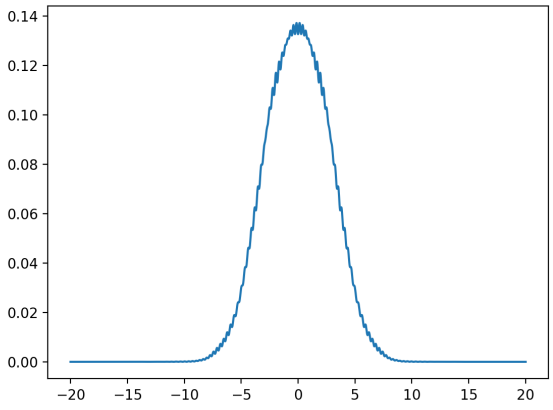


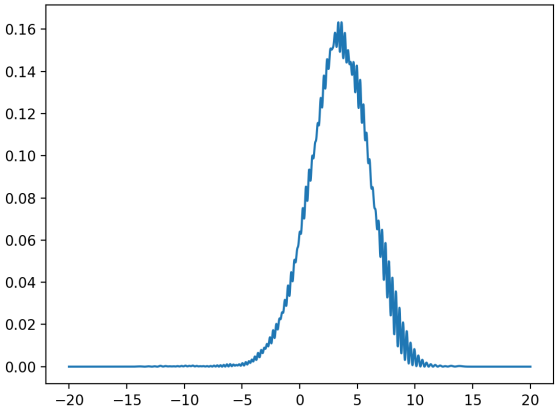
This is a gif for the case $\epsilon = -1$



```python
def spatial_wavefunctions2(N, x, epsilons):
    x[x == 0] = 1e-200
    PSI_ns = []
    for n in range(N):
        psi_n = cpsi_blank(n, x)
        PSI_ns.append(psi_n)
    PSI_ns = np.array(PSI_ns)
    np.save(f"PSI_ns.npy", PSI_ns)

    eigenstates = []
    for i, ϵ in enumerate(epsilons):
        c = coefficients[i]
        for j in range(N):  # for each eigenvector
            d = c[:, j]
            psi_jx = np.zeros(x.shape, complex)
            for n in range(N):  # for each H.O. basis vector
                psi_jx += d[n] * PSI_ns[n]
            plt.plot(x, abs(psi_jx) ** 2)
            print(f"saving {ϵ = }, {j = }")
            plt.savefig(f"some_wavefunctions/wavefunction_{ϵ}_{j:03d}.png")
            plt.clf()
            eigenstates.append(psi_jx)

plt.clf()
N = 100
epsilons = [-0.7]
xs = np.linspace(-20, 20, 2048)
spatial_wavefunctions2(N, xs, epsilons)
```

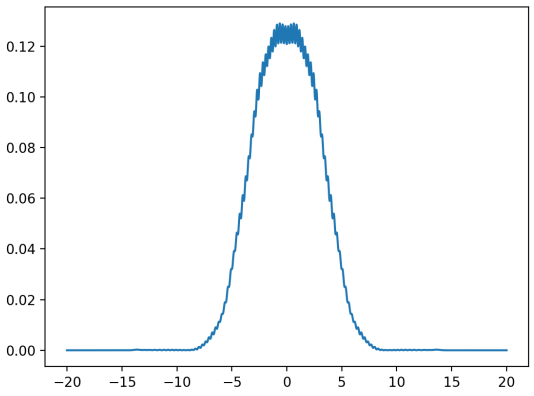
Whoops, I corrected a bug with my indentation, and the labelling of my files. Also, I am now testing a different epsilon value because $\epsilon = -1$ is a bit of a special case and I want something less infinite-order-exceptional-point-y.
These are two eigenstates for the case $\epsilon = -0.7$

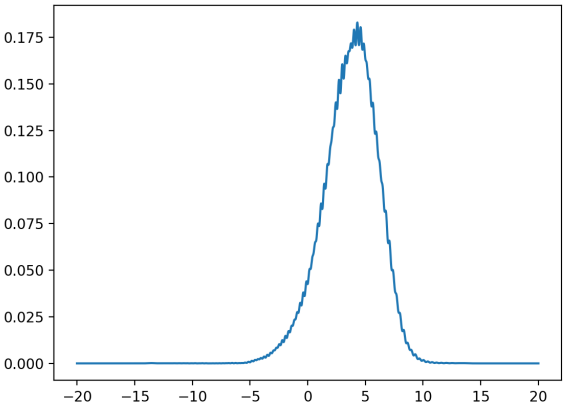




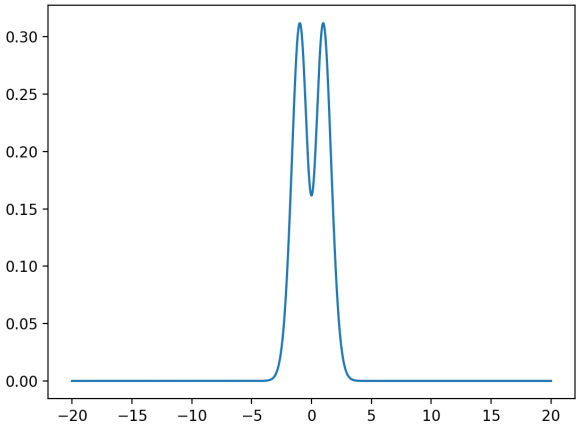
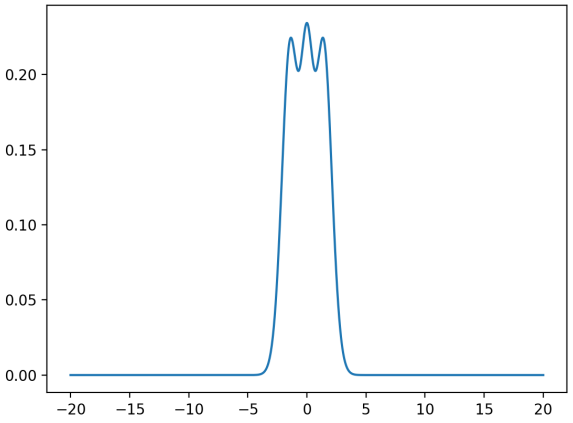
These randomly chosen eigenstates don't look orthogonal!

What about I run the same code but for $\epsilon = -0.4$? Just so I can see the states for an epsilon value that is on the other side of the exceptional point.



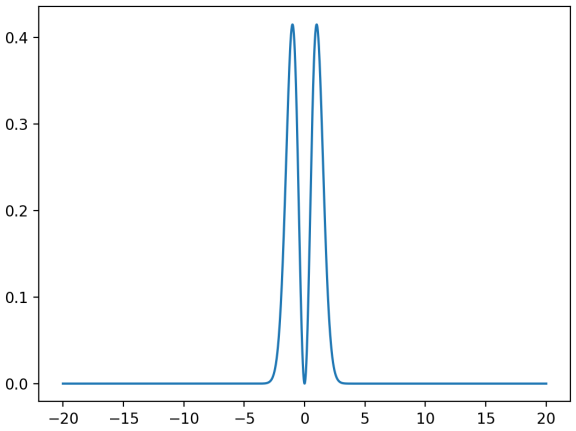
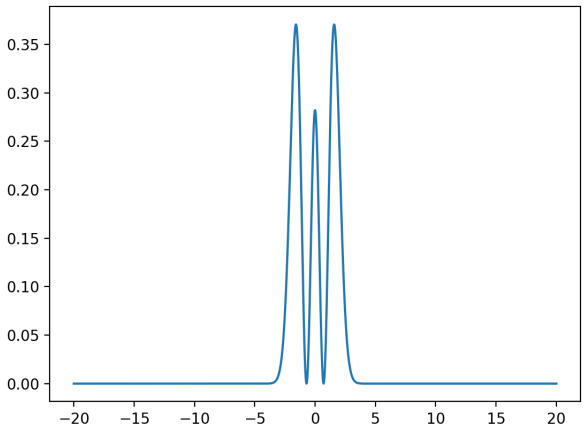


Bit of aliasing but these states also don't look orthogonal

HEYYYY I think I know these guys…They look like my old first and second excited state buddies…

Time for a sanity check. HERE I plotted the first and second excited states of the $\epsilon = 0$

Everything looks like it's working fine.

The end goals of this exercise are:

1. Verify if the eigenvectors are orthogonal
   IT DOESN'T LOOK LIKE THEY ARE ORTHOGONAL IN GENERAL…
   But I could be wrong because Im doing an inner product in my head and that's basically my "orthogonality test".
   So take this diagnosis with a grain of salt
2. Observe the coalescent behaviour of the states at the EPs

This code is too exhaustive.., there are too many states here… I am only interested in the (possibly 11) lowest energy states for each ϵ.
Therefore I need to come up with a way to order the states so I can be sure of their correspondence to the eigenvalues I already sorted.

16/05/2021

Jesper's suggestion

"...I was just wondering if for your presentation you could create some animations that illustrate how the wave functions change as you move along the solutions in your energy vs epsilon diagram? For instance, I would be very curious to see how a state wave function continuously becomes another excited state. I'm still not sure that I see how a node would appear in the wave function, does this happen at precisely the exceptional point?"

Therefore, I want to check out how the eigenstates change as ϵchanges. I want to see how the eigenstates behave as I cross the EP ϵ.
Here I want to sort the eigenstates using the second and third elements of the eigenvalue axis.

This is tricky… and I don't know how to do it yet.

I do this because I know that these eigenstates certainly coalesce at the EP.
To plot my eigenstates and obtain a sensical plot when I combine the eigenstates in a plot, I must normalise the states by

$$\psi_j(x_i) = \frac{\psi_j(x_i)}{\sqrt{\sum_i |\psi(x_i)|^2 \Delta x}},$$

Where $\psi(x_i)$ are the individual points forming the wavefunction.
Since the eigenstates might be returned with an arbitrary phase. In order to avoid ugly discontinuities in my plot frames I want to eliminate these arbitrary phases by multiplying the state by the exact opposite phase factor.

$$\psi_j(x_i) = \psi_j(x_i) e^{-iArg(\psi_j(x_i)}$$

Week 11 (17/05/21 –23/05/21)

Aims:

1. ~~Prepare TALK (12 minutes)~~
    a. Due week 11 for tqm group presentation
    b. Due Week 12 Thursday 27/05 9:45am

Tasks:

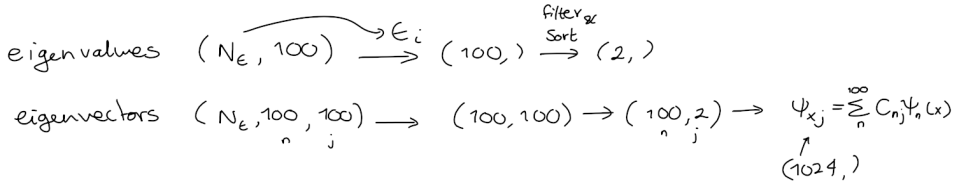
1. ~~Figure out how to plot the first and second excited states~~
2. ~~Make animation~~

18/05/2021

```python
##################### eigenvalues & eigenvectors ##########################

Energies_2 = np.load("Energies_unbroken.npy")

def linear_algebra(epsilons):

    eigenvalues_list = []
    eigenvectors_list = []
    for i, ϵ in enumerate(epsilons):
        matrix = np.load(f'matrices/matrix_{i:03d}.npy')
        eigenvalues, eigenvectors = linalg.eig(matrix)
        eigenvalues_list.append(eigenvalues)
        eigenvectors_list.append(eigenvectors)

        ϵ_list = np.full(len(eigenvalues), ϵ)

    return np.array(eigenvalues_list), np.array(eigenvectors_list)
```

Chris taught me a neat way to use boolean arrays in my indexing.
This allows me to conditionally select the elements in an array that satisfy the conditions I may want.
np.argsort() is great because it allows me to extract the indexes of my filtered eigenvalue array, and with these indexes I can sort my filtered eigenvectors so they map to the filtered eigenvalues.

My algorithm follows this path:

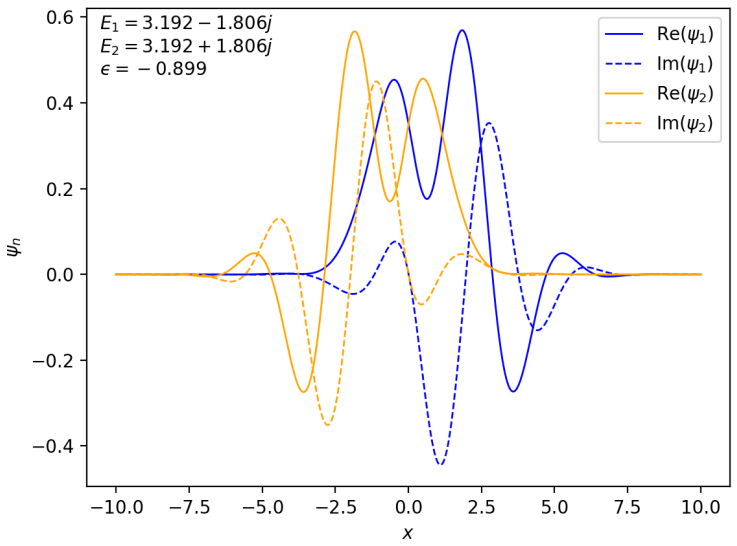



```python
##################### sorting Eigenvectors ################################

def filtering_and_sorting(evals, evects):
    mask = (0 < evals.real) & (evals.real < 20)
    # print(mask)
    evals = evals[mask]
    evects = evects[:, mask]

    # sorting
    order = np.argsort(np.round(evals.real, 3) + np.round(evals.imag, 3) / 1e6)
    # print(order)
    evals = evals[order]
    evects = evects[:, order]

    # print(evals)

    return evals[1:3], evects[:, 1:3]
```

```python
##################### Eigenvectors plot ################################

def spatial_wavefunctions(N, x, epsilons, evals, evects):
    # calculating basis functions
    x[x == 0] = 1e-200
    PSI_ns = []
    for n in range(N):
        psi_n = cpsi_blank(n, x)
        PSI_ns.append(psi_n)
    PSI_ns = np.array(PSI_ns)

    # plt.plot(x, PSI_ns[1])
    # plt.show()
    # ass
    np.save(f"PSI_ns.npy", PSI_ns)


    for i, ϵ in enumerate(epsilons):

        eigenvalues, c = filtering_and_sorting(evals[i], evects[i])

        if c.shape[1] < 2:
            print(i, "continuing")
            continue
        eigenstates = []
        for j in range(2):
            # print(len(evects))
            d = c[:, j]
            psi_jx = np.zeros(x.shape, complex)
            # for each H.O. basis vector relevant to the filtered and sorted eigenvectors
            for n in range(N):
                psi_jx += d[n] * PSI_ns[n]
                # normalise
                psi_jx /= np.sqrt(np.sum(abs(psi_jx) ** 2 * delta_x))
                # impose phase convention at ~ x = 0
                psi_jx *= np.exp(-1j * np.angle(psi_jx[Nx // 2]))

            eigenstates.append(psi_jx)

        fig, ax = plt.subplots()
        plt.plot(x, np.real(eigenstates[0]), "-", color='blue', linewidth=1, label=fr"Re($\psi_1$)")
        plt.plot(x, np.imag(eigenstates[0]), "--", color='blue', linewidth=1, label=fr"Im($\psi_1$)")
        plt.plot(x, np.real(eigenstates[1]), "-", color='orange', linewidth=1, label=fr"Re($\psi_2$)")
        plt.plot(x, np.imag(eigenstates[1]), "--", color='orange', linewidth=1, label=fr"Im($\psi_2$)")
        plt.legend(loc="upper right")
        plt.xlabel(r'$x$')
        plt.ylabel(r'$ \psi_{n}$')
        textstr = '\n'.join((
            fr'$E_1 = {eigenvalues[0]:.03f}$',
            fr'$E_2 = {eigenvalues[1]:.03f}$',
            fr'$ϵ = {ϵ:.03f}$'
            ))

        # place a text box in upper left in axes coords
        ax.text(0.02, 0.99, textstr, transform=ax.transAxes, verticalalignment='top')
        # plt.show()
        plt.savefig(f"spatial_wavefunctions/wavefunction_{i:03d}.png")
        plt.clf()
    return eigenvalues, eigenstates
```

```python
####################### Function calls ###################################

N = 100
Nϵ = 100
Nx = 1024
epsilons = np.linspace(-1.0, 0, Nϵ)
xs = np.linspace(-10, 10, Nx)
delta_x = xs[1] - xs[0]

evals, evects = linear_algebra(epsilons)

spatial_wavefunctions(N, xs, epsilons, evals, evects)
```

Here is my animation of the wavefunction behaviour of the first two excited states. I decided to not include values lower than $\epsilon = -0.8$. Because they are not actually representative of the wave functions due to the broken appearance of my numerics below this epsilon value.
The wave functions also look like they have some discontinuous steps at certain epsilon values but have no idea why that is.
In addition, the final wave functions look like they aren't decaying to zero and I think that might be because of the scaling of the HO basis states.



20/05/21
Today I had my first in-person meeting with the TQM group.
In addition I presented my research project to them as a practice presentation for the event next week.

The group asked me a lot of questions and gave me really good feedback.

I've modified my presentation slides according to most of the suggestions I received.

Also, I want to try to fix my latest wave functions animation.
Maybe I can try using the correctly normalised HO states for my equation since I didn't use the natural length and energy scales as I required in my TISE.

Week 12 (24/05/21 –30/05/21)

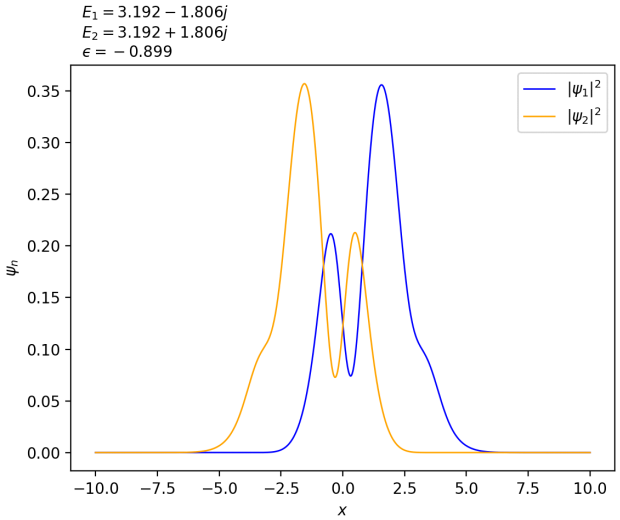
Aims:

Practice practice practice TALK (get to 12 minutes)
Due Week 12 Thursday 27/05 10am

Tasks:

1. ~~Understand why animation of eigenstates has shifted 0 value and if this is leading to deformed states (?)~~
2. ~~Fix animation if necessary~~

This is the eigenstates animation using only the absolute value squared eigenstates



This animation is a lot clearer when crossing the EP epsilon. Since the eigenstates get transformed more neatly from complex conjugate pair eigenenergies to the first and second energy level eigenstates of the systems.

The issue that I see here is that there is a very strong asymmetry in the first excited state.

24/05/21

The length scale of the HO eigenstates

I am pretty certain that I made a mistake previously when considering the natural length scale of my problem.
Previously I thought that
$\hbar = 1, m = \frac{1}{2}, \omega = 1$
The H.O. eigenstates should be:

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!}} \left(\frac{1}{2\pi}\right)^{1/4} e^{\frac{-x^2}{4}} H\left(\frac{1}{\sqrt{2}}x\right)$$

My non-Hermitian Hamiltonians

$$\hat{H} = -\frac{d^2}{dx^2} + \hat{x}^2 (i\hat{x})^{\epsilon} \qquad (1)$$

Harmonic oscillator Hamiltonian

$$\hat{H}_{HO} = \frac{\hat{p}^2}{2m} + \frac{1}{2} m\omega^2 \hat{x}^2 \qquad (2)$$

if (1) = (2):

$$-\frac{d^2}{dx^2} + \hat{x}^2 (i\hat{x})^{\epsilon} = \frac{\hat{p}^2}{2m} + \frac{1}{2} m\omega^2 \hat{x}^2$$

$$-\frac{d^2}{dx^2} + \hat{x}^2 (i\hat{x})^{\epsilon} = \frac{1}{2m}\left(-i\hbar \frac{d}{dx}\right)^2 + \frac{1}{2} m\omega^2 \hat{x}^2$$

$$-\frac{d^2}{dx^2} + \hat{x}^2 (i\hat{x})^{\epsilon} = \frac{-\hbar^2}{2m} \frac{d^2}{dx^2} + \frac{1}{2} m\omega^2 \hat{x}^2$$

in my problem I let $\hbar = 1$ & $m = \frac{1}{2}$

$$\therefore -\frac{d^2}{dx^2} + \hat{x}^2 (i\hat{x})^{\epsilon} = -\frac{d^2}{dx^2} + \frac{1}{4}\omega^2 \hat{x}^2$$

$$\therefore (i\hat{x})^{\epsilon} = \frac{1}{4}\omega^2$$

if $\omega = 2$ then $(i\hat{x})^{\epsilon} = 1$

Fortunately this error didn't actually get through my computations, since I ended up using the regular expressions for the HO eigenstates (not scaled in x for my TISE)

$$\psi_n(x) = \frac{1}{\sqrt{2^n\, n!}} \cdot \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \cdot e^{-\frac{m\omega x^2}{2\hbar}} \cdot H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right), \qquad n = 0, 1, 2, \ldots.$$

From wikipedia: Quantum harmonic oscillator: Natural length and energy scales

Natural length and energy scales [edit]

The quantum harmonic oscillator possesses natural scales for length and energy, which can be used to simplify the problem. These can be found by nondimensionalization.

The result is that, if *energy* is measured in units of $\hbar\omega$ and *distance* in units of $\sqrt{\hbar/(m\omega)}$, then the Hamiltonian simplifies to

$$H = -\frac{1}{2}\frac{d^2}{dx^2} + \frac{1}{2}x^2,$$

while the energy eigenfunctions and eigenvalues simplify to Hermite functions and integers offset by a half,

$$\psi_n(x) = \langle x \mid n\rangle = \frac{1}{\sqrt{2^n n!}}\, \pi^{-1/4} \exp(-x^2/2)\, H_n(x),$$

$$E_n = n + \tfrac{1}{2}\,,$$

where $H_n(x)$ are the Hermite polynomials.

This means that in the natural scale of the HO states : $\frac{m\omega}{\hbar} = 1$

If I go ahead and use my assumed values for $\hbar = 1$, $m = \frac{1}{2}$, $\omega = 2$

Then $\frac{m\omega}{\hbar} = 1$ is satisfied.

Therefore I can conclude that I require the unscaled HO basis in order to solve my TISE (as it is in the screenshot above)

This means that the strange shift of zero in my original animation for the first and second excited states of my parametric family of Hamiltonians remains an unsolved mystery.

26/05/21

Bug fix

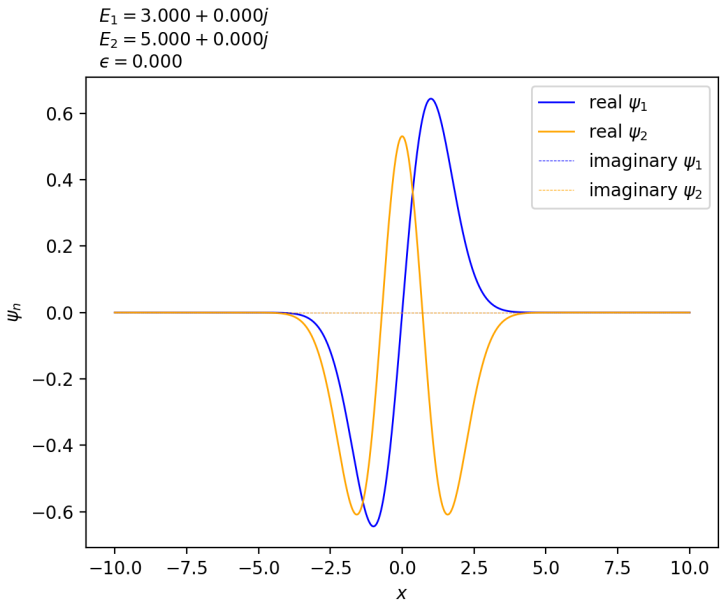
I think I finally located the source of my asymmetrical states!
I had a simple bug. I was normalising and imposing my phase convention too early in the n loop

```
    for n in range(N):
        psi_jx += d[n] * PSI_ns[n]
        # normalise
        psi_jx /= np.sqrt(np.sum(abs(psi_jx) ** 2 * delta_x))
        # impose phase convention at ~ x = 0
        psi_jx *= np.exp(-1j * np.angle(psi_jx[Nx // 2]))

    eigenstates.append(psi_jx)
```

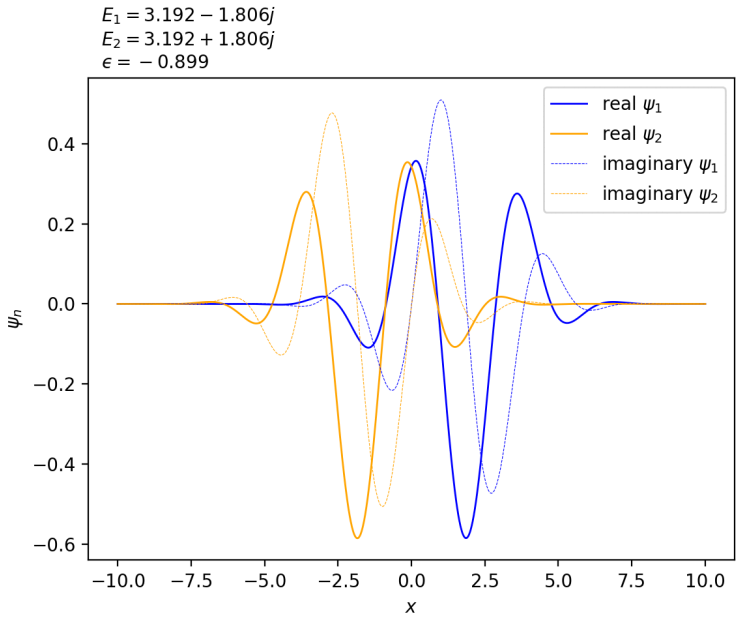
--->

```
    for n in range(N):
        psi_jx += d[n] * PSI_ns[n]
    # normalise
    psi_jx /= np.sqrt(np.sum(abs(psi_jx) ** 2 * delta_x))
    # impose phase convention at ~ x = 0
    psi_jx *= np.exp(-1j * np.angle(psi_jx[Nx // 2]))

    eigenstates.append(psi_jx)
```

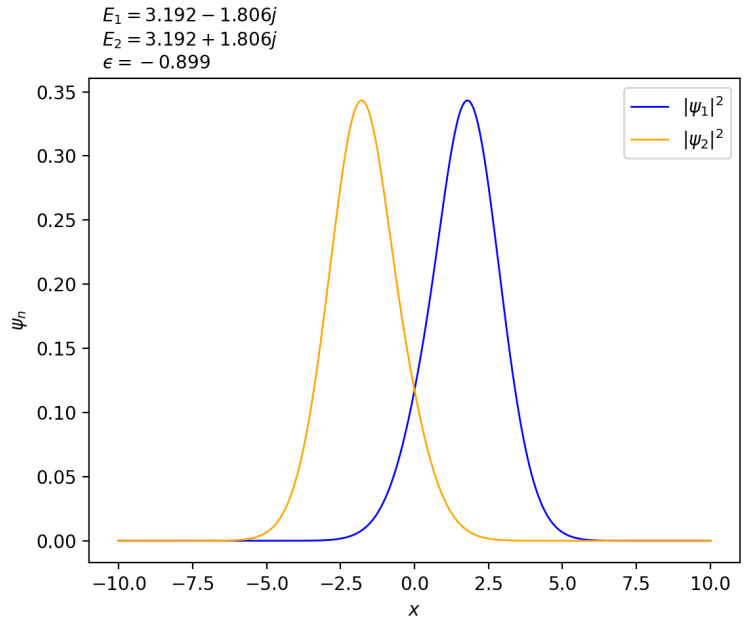
YAY success! I am finally seeing the HO eigenstates when epsilon is zero.



This is the animation of the wave functions as epsilon changes



This is the animation of the probability densities of the wavefunctions



Week ∞ (06/06/21 –12/06/21)

Aims:

1. Finish Report 3
    a. Orthogonality of eigenfunctions
        i. Check inner products of wavefunctions for cases $\epsilon = \ -0.7$ and $\epsilon = \ -0.4$

Tasks:

1. ~~Submit Logbook~~
2. ~~Submit Report 3~~

09/06/21

Bibliography

ANU. “Playing quantum billiards with light and matter.” *phys.org*, Science X Network, 13th October 2015, https://phys.org/news/2015-10-quantum-billiards.html. Accessed 16th March 2021.

Bauke, Heiko. “Calculating Hermite functions.” *Number Crunch - A computational science blog*, 21st August 2014, https://www.numbercrunch.de/blog/2014/08/calculating-the-hermite-functions/. Accessed 7th May 2021.

Bender, Carl M., et al. “Behaviour of eigenvalues in a region of broken-PT symmetry.” *Physical Review A*, vol. 95, no. 5, 2017, http://dx.doi.org/10.1103/PhysRevA.95.052113. Accessed 10 May 2021.

Bender, C. M. “Making sense of non-Hermitian Hamiltonians.” *Reports on Progress in Physics*, vol. 70, p. 947.

Gao, T., et al. “Observation of non-Hermitian degeneracies in a chaotic exciton-polariton billiard.” *Nature*, vol. 526, (2015), pp. 554–558.

Moiseyev, N. *Non-Hermitian Quantum Mechanics*. Cambridge University Press, 2011.

Press, W. H., et al. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3 ed., 40 W. 20 St. New York, NY United States, Cambridge University Press, 2007. *Numerical Recipes Bookreader Home*, http://numerical.recipes/book/book.html. Accessed April 2021.

Sorrell, M. “Complex WKB Analysis of a PT Symmetric Eigenvalue problem.” 2007, https://arxiv.org/pdf/math-ph/0703030.pdf. Accessed 2 April 2021.