

# 0. Old Man BS?

## Contents

- 1. Thinking in another dimension
- 2. Imagine about nothing
- 3. How to imaginery something not possible to exist
- 4. Imagine another imaginery number and split it out
- Appendix - may be needed for reference
- Welcome to your Jupyter Book
- Page title
- Markdown Files
- Content with notebooks
- Notebooks with MyST Markdown
- testing jupyter-book failed using fAe examples
- testing jupyter-book ok using fFy examples
- testing jupyter-book ok using fnn examples
- testAnimated-Sinc-and-FT-example

Welcome to my little web site about imaginery number Hopeful may generate PDF/latex/book ... somehow.

This is based on [Jupyter Book](#). Just like everything in here, everything is just a result of my hobby and learning.

The logo is from [the Wikipedia page on Imaginery Number](#); copyright are theirs.

And later "chapters" is just the sample pages from the book of Jupyter Book. Keep them as it is my testing whether it is the setup issue (if even they cannot be presented) or it is on my page content. Sorry, incomplete, draft, ...

[Skip to main content](#)

# In case of doubt ... it is just a hobby not for "production"

I am not sure anything is correct here. As said just my hobby and hence if you find anything error, mistake, omission, ... etc., please do alert me.

Check out the content pages here to see more. Just do not not trust it is right. Take them at best as a starting point of your investigation. If I can make you curious about the world, that is all what I aim for!!!

`` `{tableofcontents}

`` `

`` `{contents}

:local:

`` `

## 1. Thinking in another dimension

Thinking in another dimension

... even it does not exist  
... even if it is impossible to think about

My aim is not to "teach" maths  
or purely introduce you to any numbering system

My objective is to  
try to see a way to get one to think outside the box  
or to see another dimension even if it does not exist

[Skip to main content](#)

And even if not agree ...

If not, at least to wonder,

And to be curious ...

And even if it is impossible to imagine these

and frankly even you know deep down that dimension does not even exist!

There are many maths I think very striking when I encounter it the very first time

- Chicken and Rabbit in one cage with 4 heads and 6 legs how many ...
- what is negative number
- It is not that  $\sqrt{2}$  is irrational or not, but the proof by contradiction
- imaginary number is real and there are at least 3 kinds of them

---> real number's infinity is larger than integer number's infinity  
(technical terms is real number is uncountable)

---> any formal mathematical system which is better than arithmetic will  
have statement in it that cannot be proven or disproved in that system

(Or Should we say Mathematical Systems Always Contain Unprovable Truths)  
And even worst (or brilliant?) someone proved this theorem of unprovable truth!

---> in real world, triangle is either  $\geq 180$  degree, which one then?  
only one is the truth and is real  
then why bother and what is the point of having 3 geometries  
and even worst ... find all of them are useful!!!

--> maths is useful and relevant  
But most surprising at all, is why maths is useful at all?  
Why a human endeavour can have relevance to the world?  
And what is the limit of this approach to life?

OK too much BS, let us start with a simple

--> projective geometry first

Imagine you have a circle of say radius 1 that  
sit on a line of infinite length say the real number line

From the "north pole" of that circle you draw a line towards the line  
then move left or right and map each point of the circle to the line

Can you see that each point of the circle can project to each point to the line  
(technically not the  $\pm$  infinite as you only have one north pole la  
can  $\pm$  infinite meant? let us not go that far. But even a bit short ...  
do you find it amazing?

How can it be?

[Skip to main content](#)

How can you map finite to infinity

Can you really see the world in a sand?

Or even worst as in my most dislike Fan Yin (or no feeling Indian)  
the world is all but a reflection of the pearl 因陀羅網  
Just watch one water dew on the leave you can see the whole world in it

Anyway ... next let us look at some number system as I promise to present on at least one!

```
import datetime
print(datetime.datetime.now())
```

2023-09-23 20:44:32.232003

## 2. Imagine about nothing

Leopold Kronecker, who once wrote that

--> "God made the integers; all else is the work of man."

- a key and very dominant mathematicians who object irrational number, pi, set theory
- It was finally proved that pi is transcendental number ...
- but as he said what is the point, they do not "exist" ! Illusion!!!
- (In a way he is not alone,  
a lot of mathematicians are also not very uncomfortable with negative number

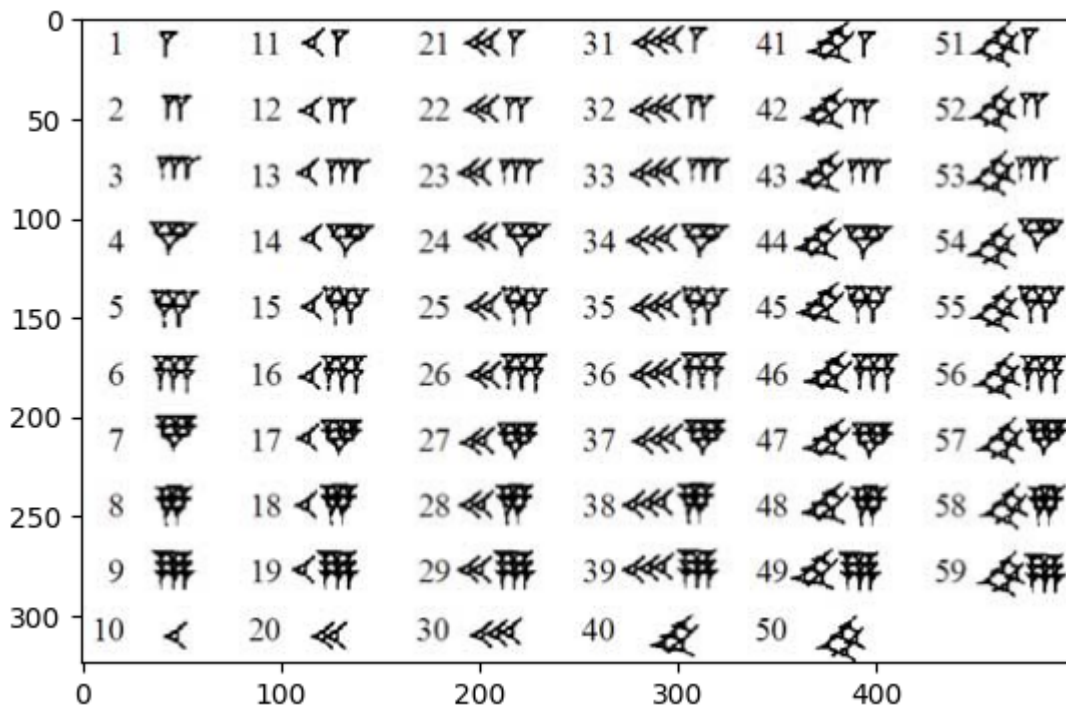
--> Let us do zero first and then imaginary number

```
import os
print(os.getcwd())
```

/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno\_book1/imgnobk1

```
import lib.main.a0_babylon_pos_0
```

```
lib.main.a0_babylon_pos_0.display_img(1)
```



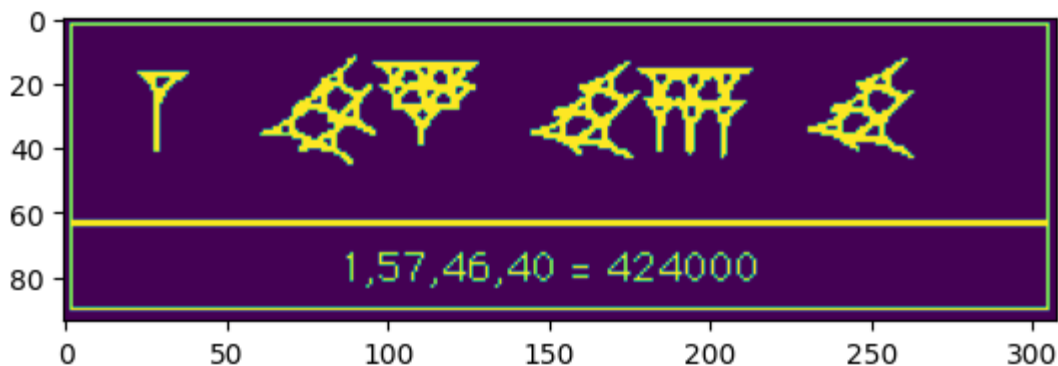
note

a) there are only 2 symbols 1 and 10

b) every number is actually by placing and counting number 1 and number 10 out;

c) reaching 59 then what ... 60 is a problem let us skip it first ;-P

```
lib.main.a0_babylon_pos_0.display_img(2)
```



d) Babylon use 60 as based and hence 1 is  $60 * 60 * \dots$

e.g. 1 here is  $60^3 \dots$

[Skip to main content](#)

The 1 above is  $1 * 60 * 60 * 60$  because its position after you read in the h

In fact even worst you cannot tell what 1 meant until you have the whole nu  
as it uses the Big Endian convention

i.e. until you reach the end of a number you do not know what is 1

It can be 1, 60,  $60*60$ ,  $60*60*...$  you do not know

(Cf the little endian for this number 40, 46, 57, 1

and after reading 40 you know it is just 40,

and after reading 46 as "nd number you know it is  $46 * 60 ...$

you do not need to wait for the whole number

or fight the egg head war of big ednian vs little endian )

The Roman does not use postional system and hence has no such issue

The only trick to remember if you see a small number earlier than a lar  
XL and LX where X is 10 and L is 50, what are the numbers ?

You do not need to know the position of X or L is you know it is always

In fact, there is no need of zero (except zero itself called nulla)

Actually Hans's does not really use a positional system

as 2,0001 is really 2 thousands and one not 2001

still have zero to sound better

f) For the issue 1574640 is 15,7 or 1,57 ... Babylon use gap

i.e. 1 gap 57 gap 46 gap 40

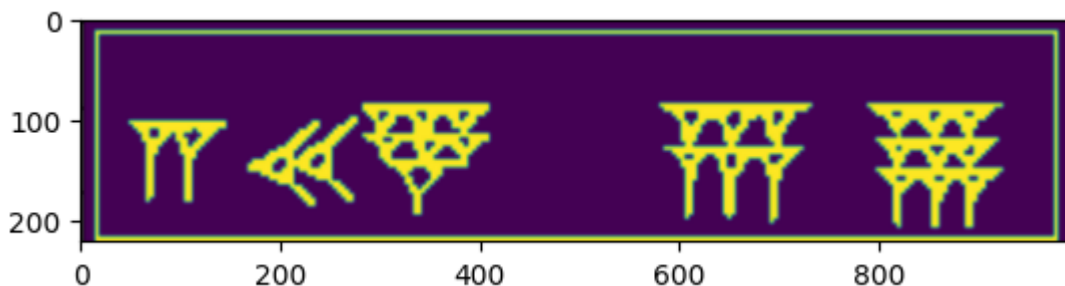
g) How about 60, 600, ...

seems to rely upon common sense?

h) But what about 601 6001 ... Big problem

The problem is that there is no zero and they use a positional system!!!

```
lib.main.a0_babylon_pos_0.display_img(3)
```



"a cuneiform tablet (actually A0 17264 in the Louvre collection in Paris)"

[Skip to main content](#)

2 ? 27      ??                  6                  ?                  9

gap,      large gap,      medium size gap

What is it?

2,27 square is 6 <0> 9 ...

2<gap>27    <very big gap>    6<larger gap>9

This is interpreted as

$2 \times 60 + 27 = 147$  \*  $147 = 21,609$  or  $6 \times 60 \times 60 + 9$

Thanks you very much for the attention!!!!

Babylon consider zero as nothingness, and does not exist  
But they need at least in their positional based number system

Even if zero does not exist  
but one may have to accept its usefulness  
and use it ... at least that is what the Babylonian did

You may think it does not exist, but your system really need it wow!!!!

IT DOES NOT EXIST  
IT IS NOT REAL  
THERE IS NOTHING THERE!!!!

Still...

Many maths and possibly many religion and philosophy are like that!!!

As said only integer really exist, all are ... (the guy is mad mathematicians btw)!

### 3. How to imaginary something not possible to exist

actually the imaginary number is unusual, but not that unusual.

Like negative number and zero it does not exist.

But it exist somehow in some way and you know it, like the zero in Babylonian system.

[Skip to main content](#)

It seems others can do this but not me

We can easily compute it analytically

$$\int_0^{\pi} \sin(x) dx = -\cos(x) \Big|_0^{\pi}$$

$$= -(\cos(\pi) - \cos(0))$$

$$= -(-1 - 1)$$

2

But let's try computing using the trapezoid rule.

It seems others can do this but not me

Fourier Transform of  $f(x)$  is  $F(k)$

$$F(k) = \mathcal{FT}\{f(x)\}$$

$$F(k) = \int_{-\infty}^{\infty} f(x) \exp(-ikx) dx$$

where  $k = \frac{2\pi}{x}$  is called the "wavenumber"

The major turning point is in the climbing of grease pole of being a maths professor!

In the video I provided (see <https://www.youtube.com/watch?v=cUzklzVXJwo>), one of the challenge at that time is the equation of

$$x^3 = 15x + 4$$

[Skip to main content](#)



Simple! 4 is an answer!!! But then we have a problem, the general equation leads to this:

$$\sqrt[3]{2 + \sqrt{-121}} + \sqrt[3]{2 - \sqrt{-121}}$$

$$\text{or } 2 + (-121)^{(1/2)^{(1/3)}} + 2 - (-121)^{(1/2)^{(1/3)}}$$

in fact this complex expression is just 4,

as it turns out it that if you treat  $\sqrt[3]{2 + \sqrt{-121}}$  as  $a + bi$  and  $\sqrt[3]{2 - \sqrt{-121}}$  as  $a - bi$  then one can find out that

$$2 + (-121)^{(1/2)^{(1/3)}} = 2 + i \text{ and } 2 - (-121)^{(1/2)^{(1/3)}} = 2 - i \text{ and together it is just 4}$$

in fact once we know 4 is a factor you can easily found out all the REAL root of this equation:

$$\text{i.e. } 4, -2 - \sqrt{3} \text{ and } -2 + \sqrt{3}$$

and the equation would be

$$(x - 4)(x - (-2 - \sqrt{3}))(x - (-2 + \sqrt{3}))$$

It is only the intermediate step use imaginary number!

(Just like later quantum mechanics, the intermediate step use imaginary number, but one can only observe real number ... a major debate leading to Coppehegan Interpretation, or just shut up and calculate!)

If imaginary numner i.e  $i^2 = -1$ , it does not exist it seems.

But could it be like the i above and the 0 in babylon above, we can treat it as existed but not ultimately.

In fact this concept also apply to the trial of Galialo!

(Just he does not accept this kind of argument. No tool argument. No just calculate...)

Our mind is bounded as we too used to the magic of the arthimetic since ancient times:

一畝≡五，一畝—為二。自此以往，巧麻不能得，而況甘凡乎！

[Skip to main content](#)

Going on from this (in our enumeration), the most skilful reckoner cannot reach (the end of the necessary numbers), and how much less can ordinary people do so see <https://ctext.org/zhuangzi/adjustment-of-controversies>

道生一，一生二，二生三，三生萬物

"The Dao produced One; One produced Two; Two produced Three; Three produced All see <https://ctext.org/dao-de-jing>

(Unlike Zhuang Zi, Laozi may also touch on older modular view of I Ching but not necessarily Zhou version; both he and Zhuang Zi likely of one dynasty One has to go back much earlier to know there might be at least 5 sources ....

+/\* can stay on natural number stated above ... may have prime/composite number

0 may or may not natural number ...

----- that is gap here as pointed out by the "mad" maths guy

Once we start to think backward then we have a lot of number/problem/issue:

- reverse of +

rotation of 180 degree or pi radian?

reflection

shift

already unnatural (0 by Indian, -ve by Chinese)

generate integer

/ reverse of \*

have rational number (by Egyptians)

(technically can generate infinite but seems not historical from this line of t (lots of strange number like finite decimal, dyadic (final binary), repeating d

but one must be careful, the reason why it is that because we try to express 1/3 as a sum of 1/10+...

/sqrt reverse of one particular type of \*

irrational especially possibly the first example of  $\sqrt{2}$

(proved by geometry then)

... algebraic irrational and a lot of others like our imaginary number

ratio and change

pi (nature unknown and only in 19th century prove to transcendental like e)

ratio of circumference and radius (radian is 2pi because of 2 pi r)

e as the growth rate or change in y is equal to y

complex number

discussed here through algebra

many strange number system now, but unlike many other systems, they ARE NUMBER SYSTEM e.g. you can +/\*//^ etc.

But we must use real number (and in fact can stay on it even if we use imaginary nu

[Skip to main content](#)

Let us SEE THE MAGIC

Note

$$1 * i = i(i^1) 1 * i * i = -1(i^2) 1 * i * i * i = -i(i^3) 1 * i * i * i * i = 1(i^5)$$

In fact one can see that every 4 operation of i comes back to 1

What is that operation

ROTATION! to be exact rotation of 90 degree or pi/2 radian

i is not on real number line and is NOT a real number no doubt,  
but could be it is a rotation of real number to another dimension

Things does not exist in the real world may not have consequences in the real world

Think

Outside the box

北冥有魚，其名為鯢。鯢之大，不知其幾千里也。  
化而為鳥，其名為鵬。鵬之背，不知其幾千里也；  
怒而飛，其翼若垂天之雲。是鳥也，海運則將徙於南冥

In the Northern Ocean there is a fish, the name of which is Kun  
– I do not know how many li in size.

It changes into a bird with the name of Peng,  
the back of which is (also) – I do not know how many li in extent.

When this bird rouses itself and flies,  
its wings are like clouds all round the sky.  
When the sea is moved (so as to bear it along),  
it prepares to remove to the Southern Ocean.  
The Southern Ocean is the Pool of Heaven.  
see <https://ctext.org/zhuangzi/enjoyment-in-untroubled-ease>

Once we accept another sky dimension we can no longer bounded by the sea

The metamorphosis of 鯢(Kun)鵬(Peng), no longer a fish but a bird!!!

Once we open up i as another dimension

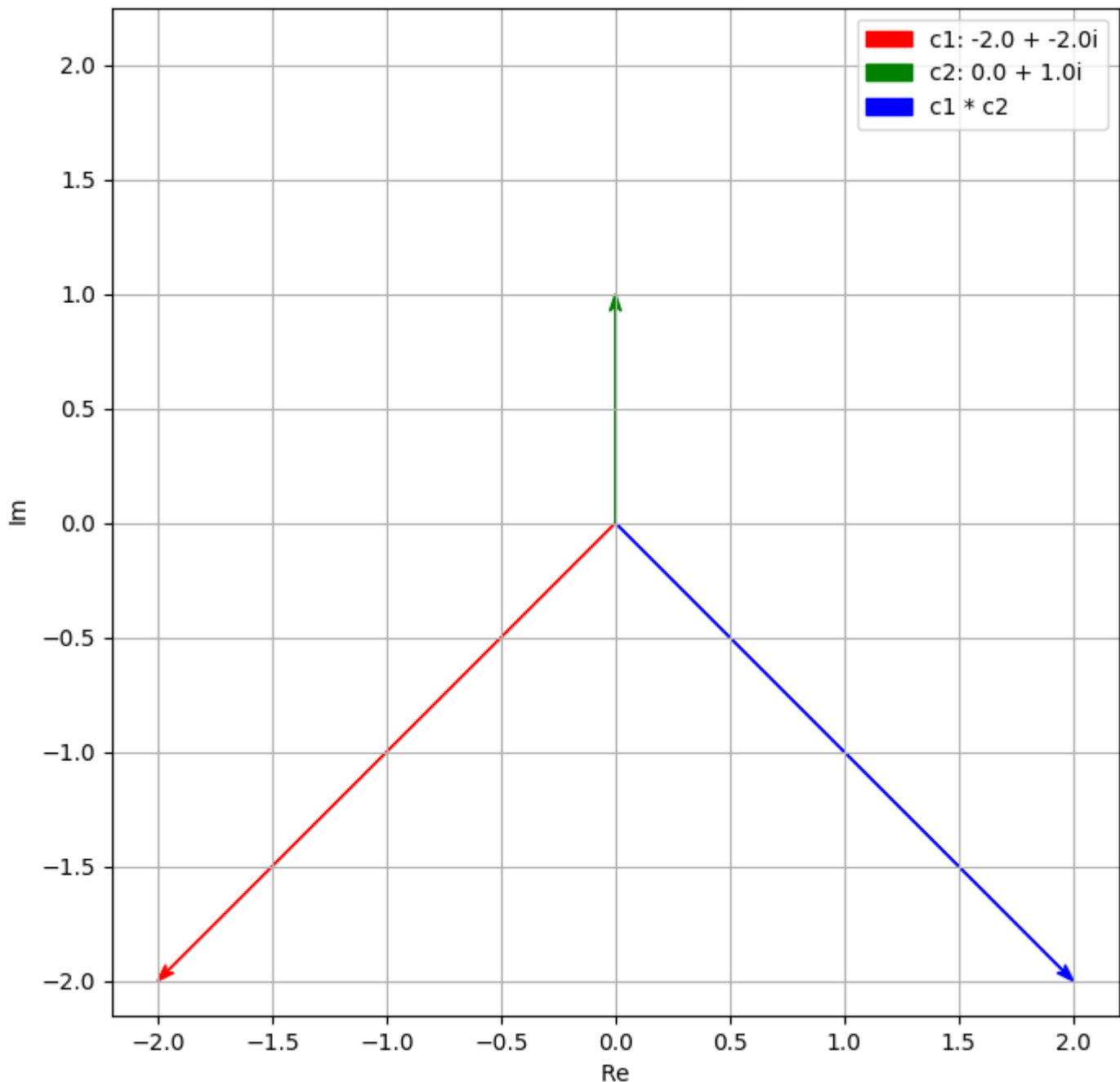
In fact it has to be 90 degree from real number dimension,

[Skip to main content](#)

The other trick now is because it comes from rotation,  
there is a lot of usefulness to it and it is now everywhere!

```
%matplotlib inline  
import lib.main.a1_i_mul_plot
```

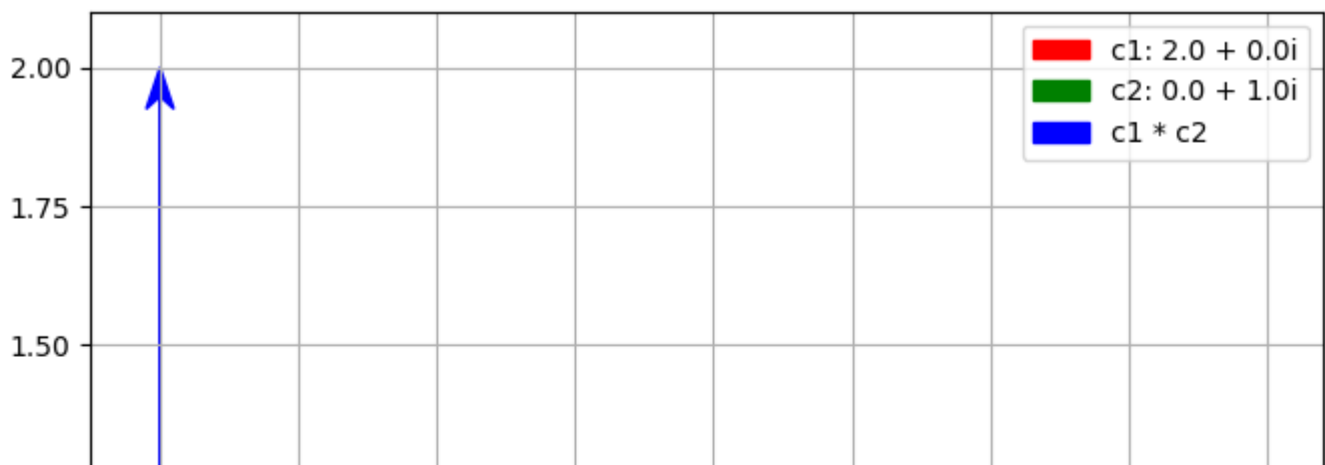
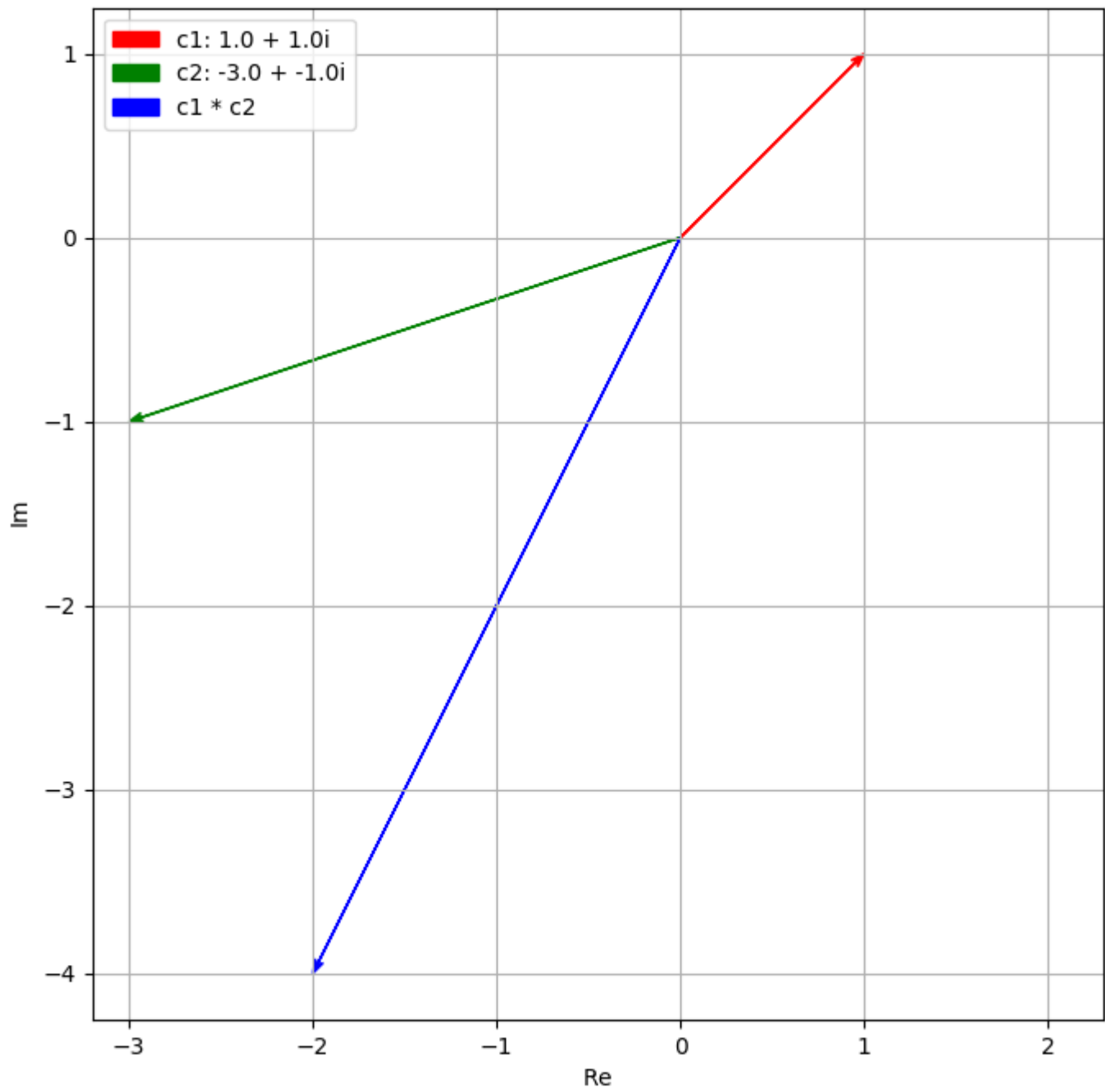
```
lib.main.a1_i_mul_plot  
not via main  
it is a rotation  
and i always rotate 90 degree or  $\pi/2$  radians
```



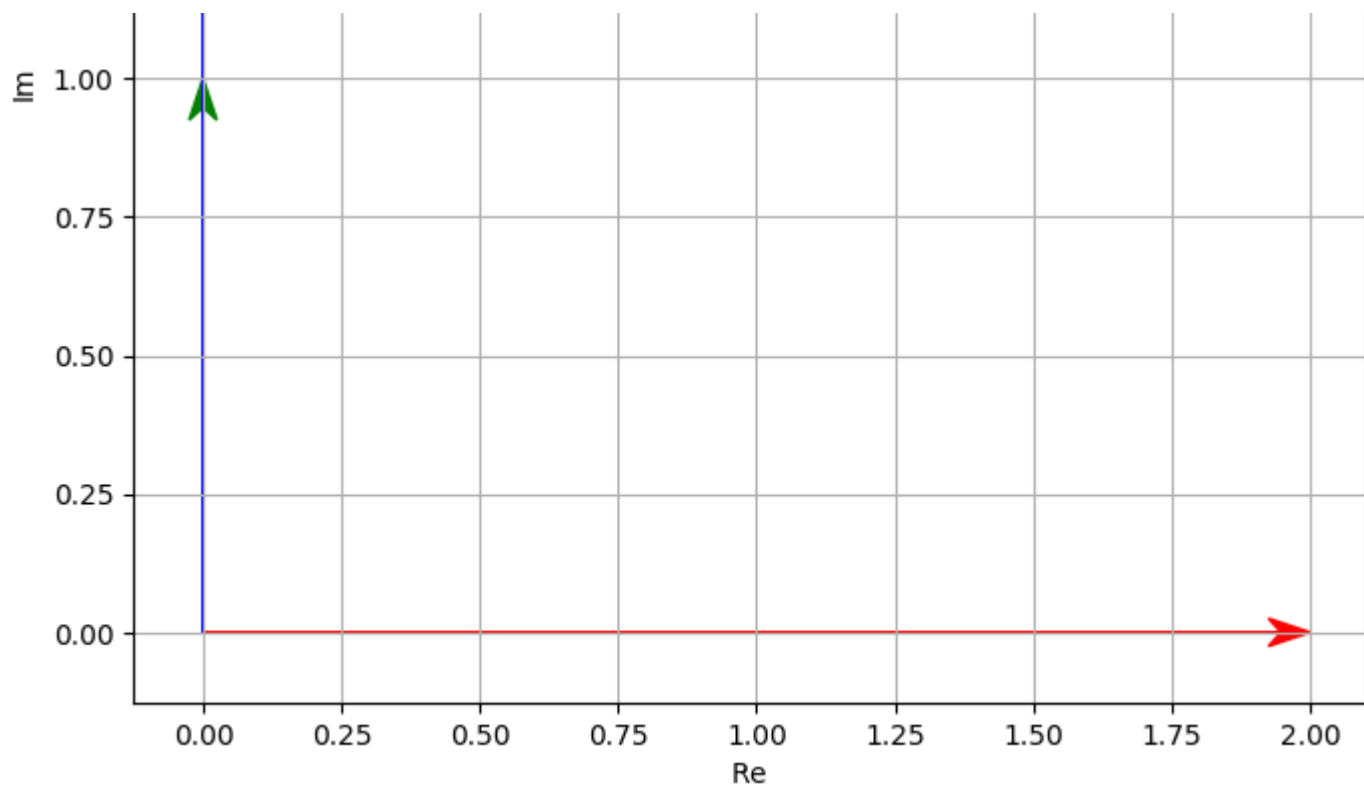
[Skip to main content](#)

```
%matplotlib inline  
lib.main.a1_i_mul_plot.some_plot()
```

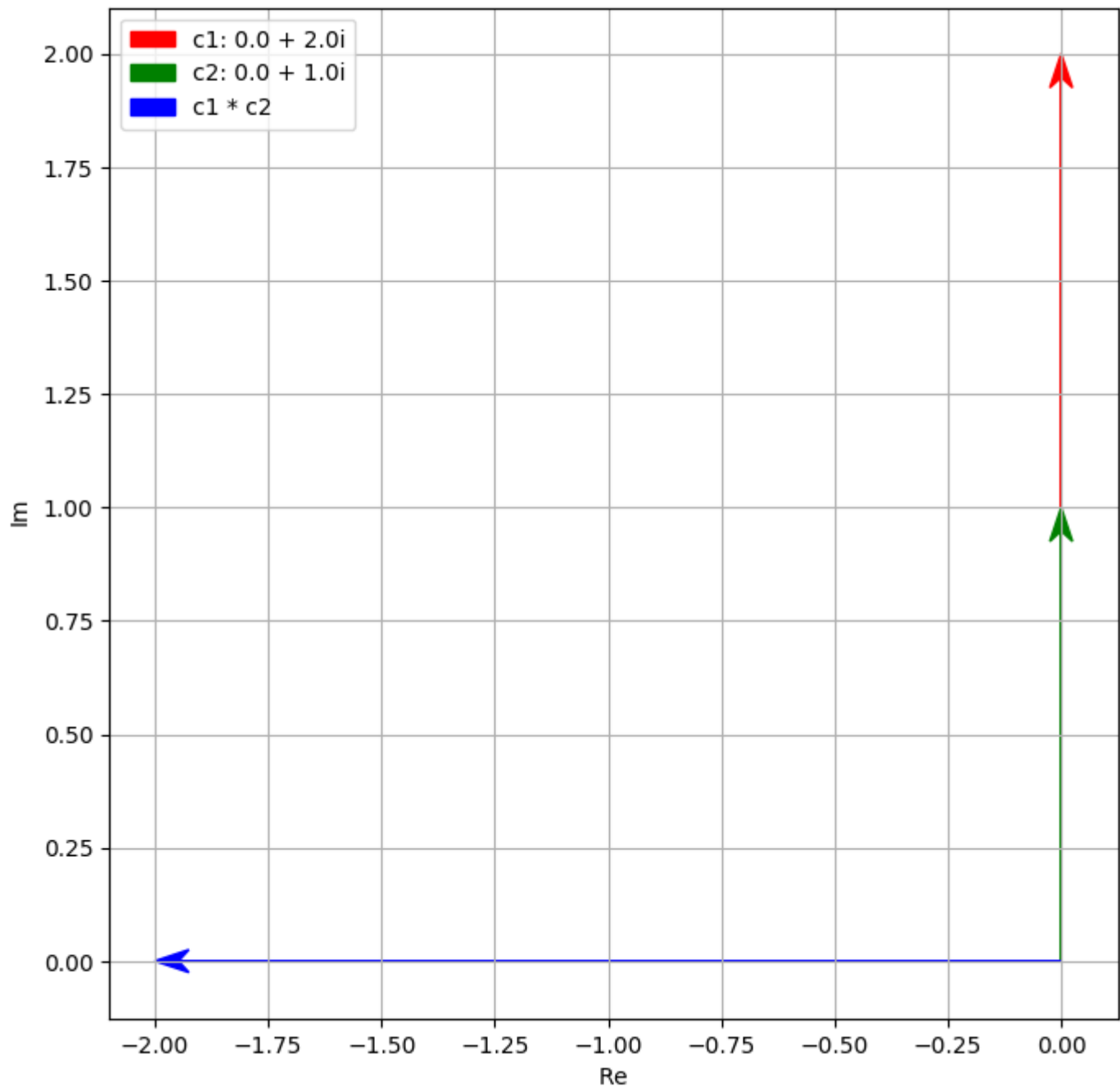
-- more example--

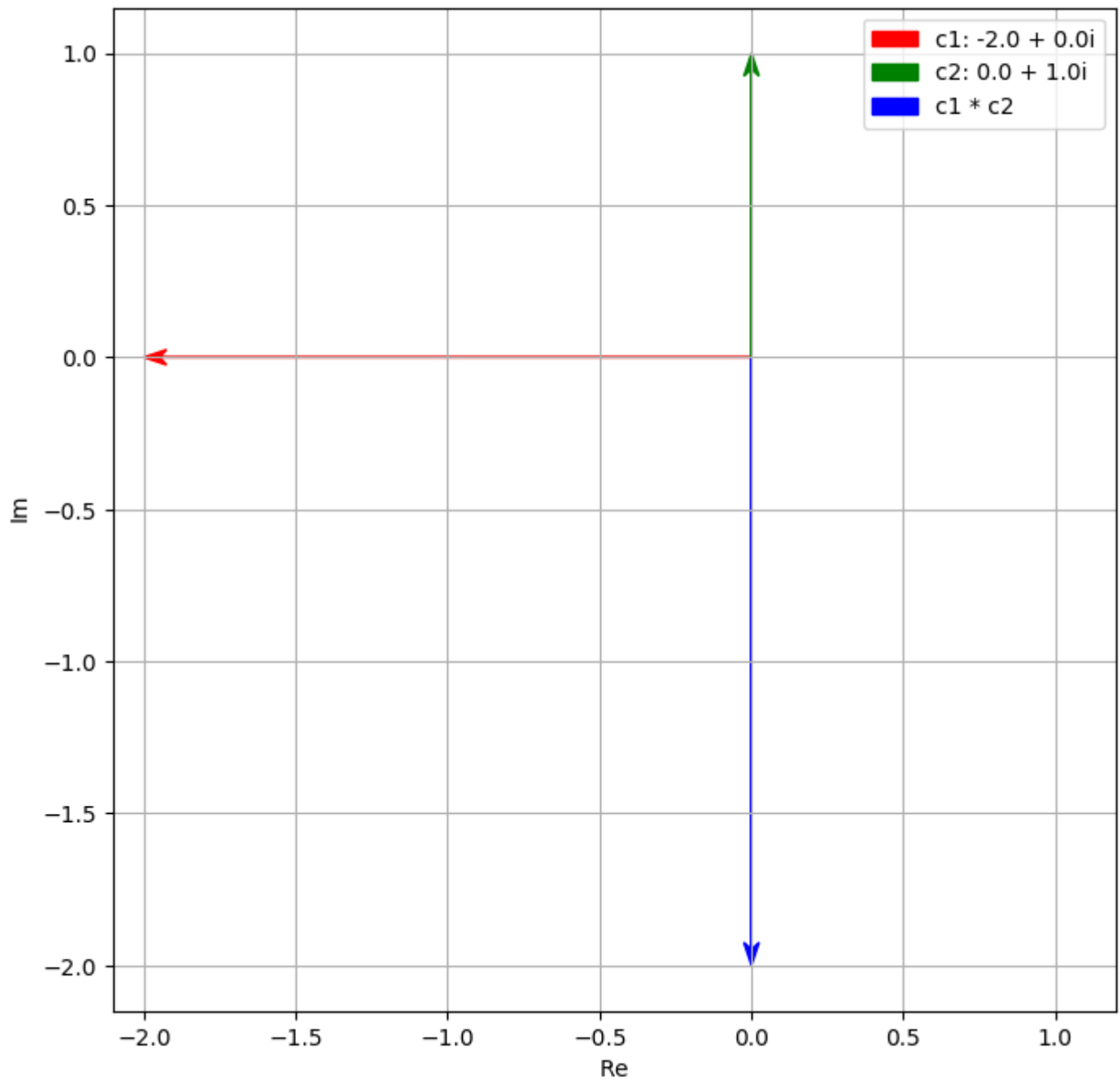


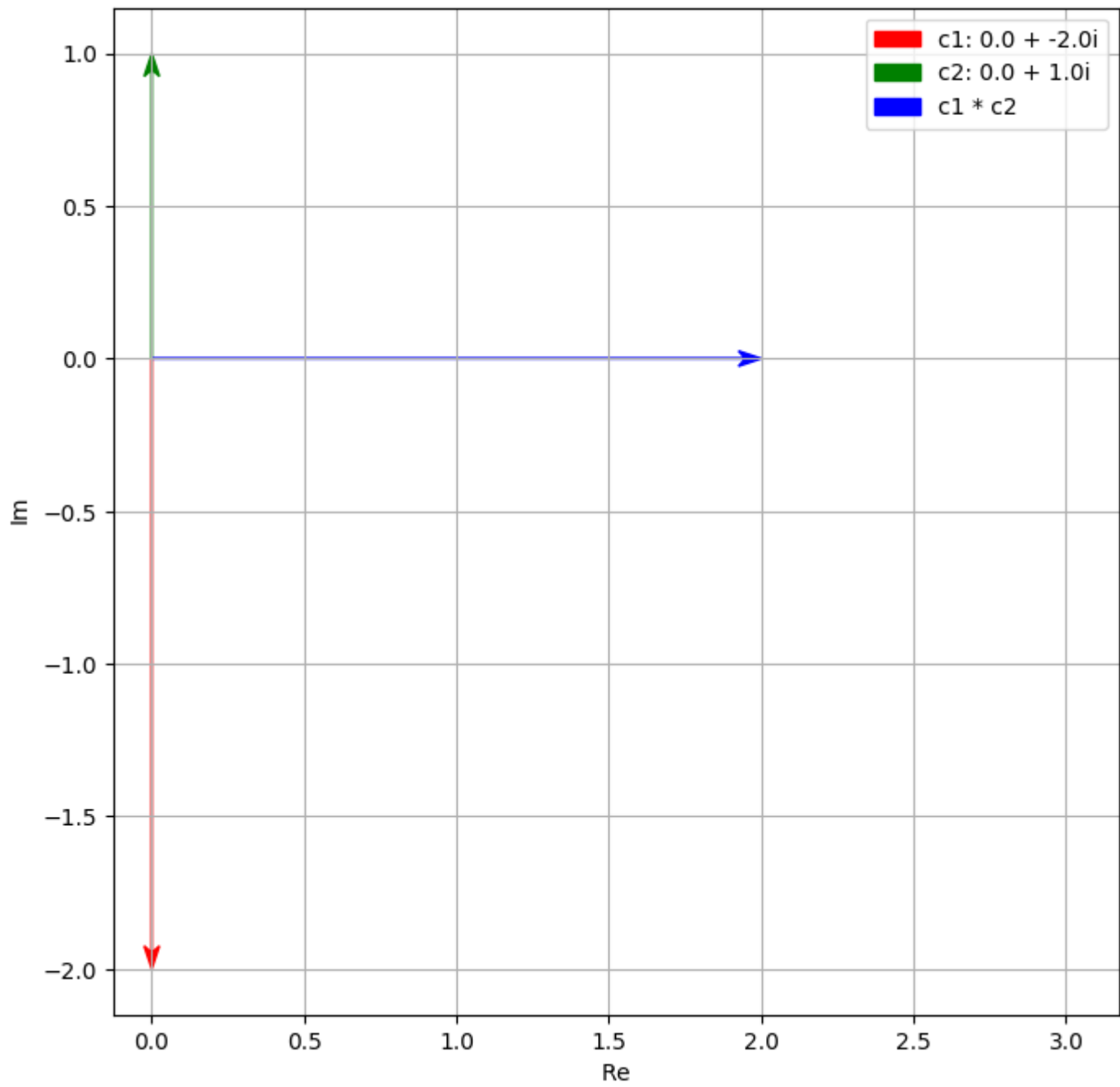
[Skip to main content](#)

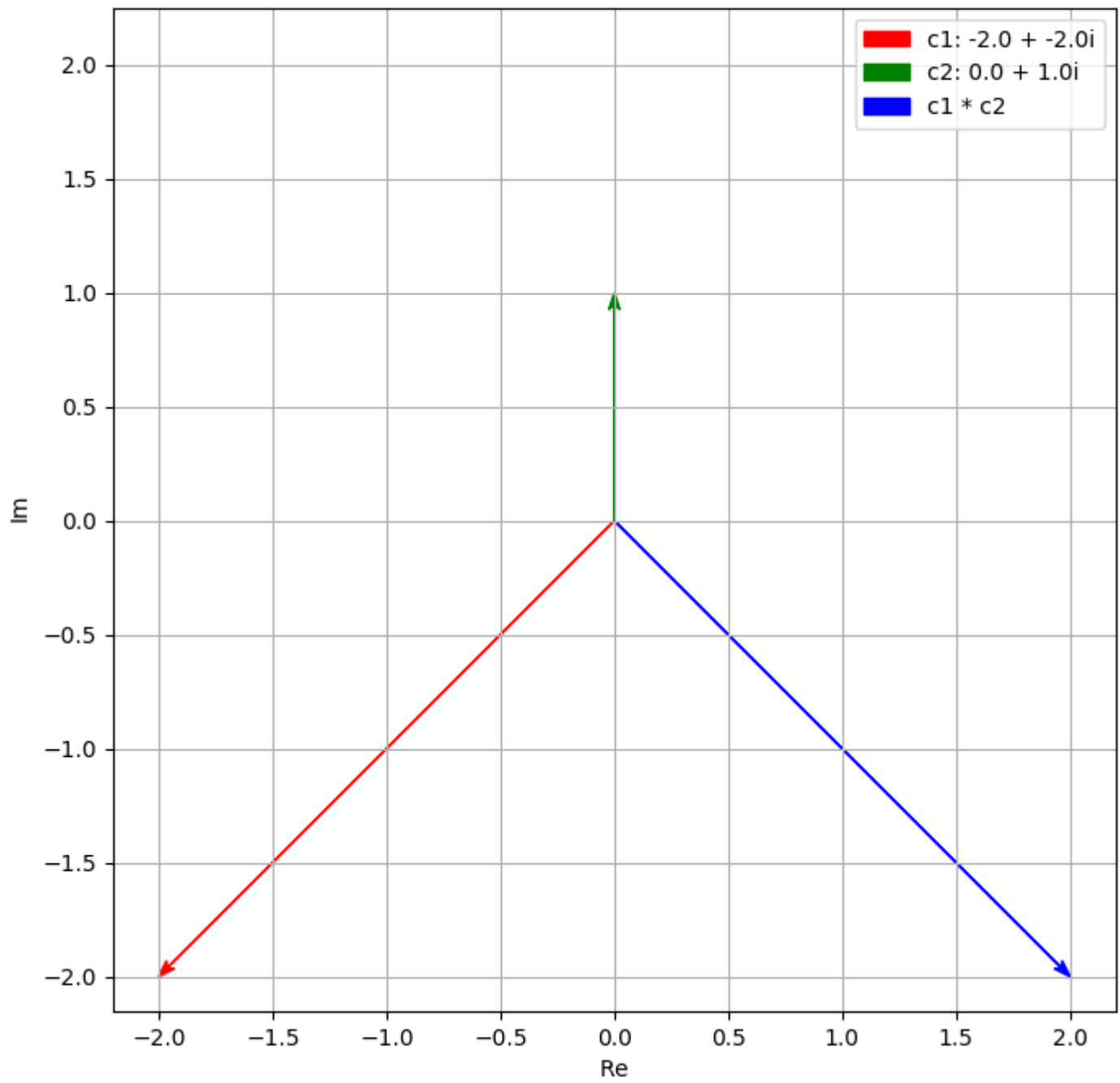


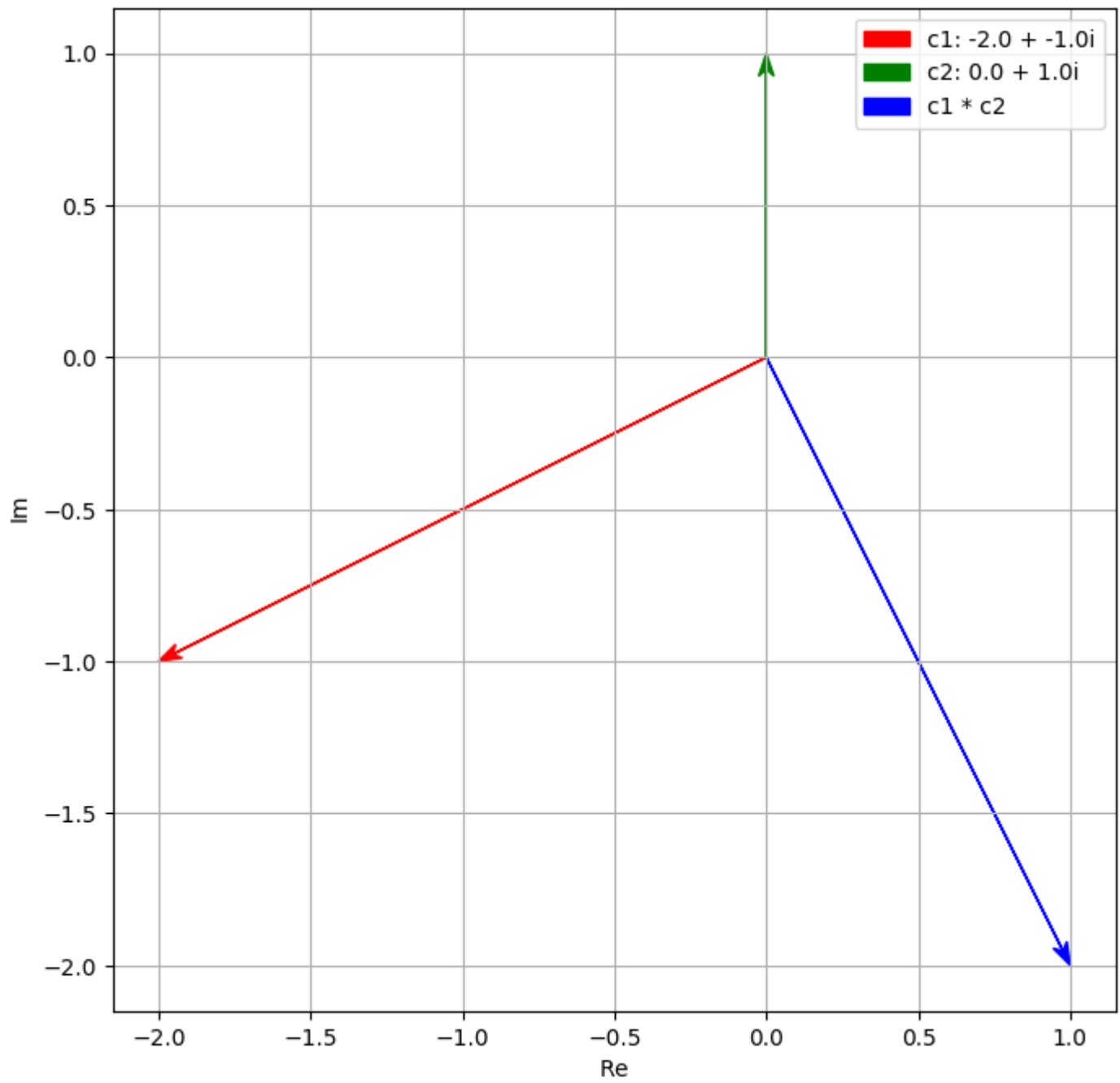


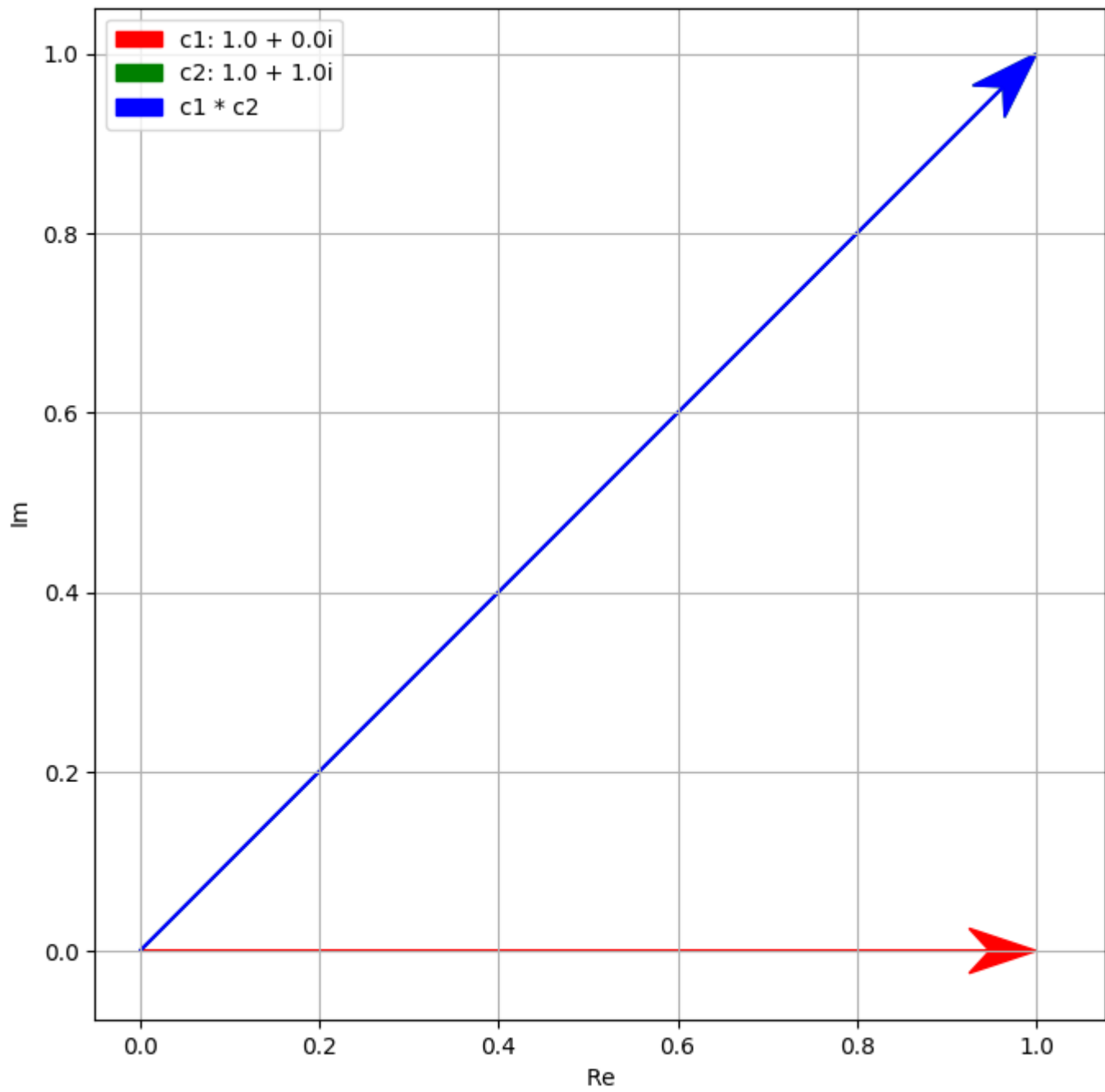


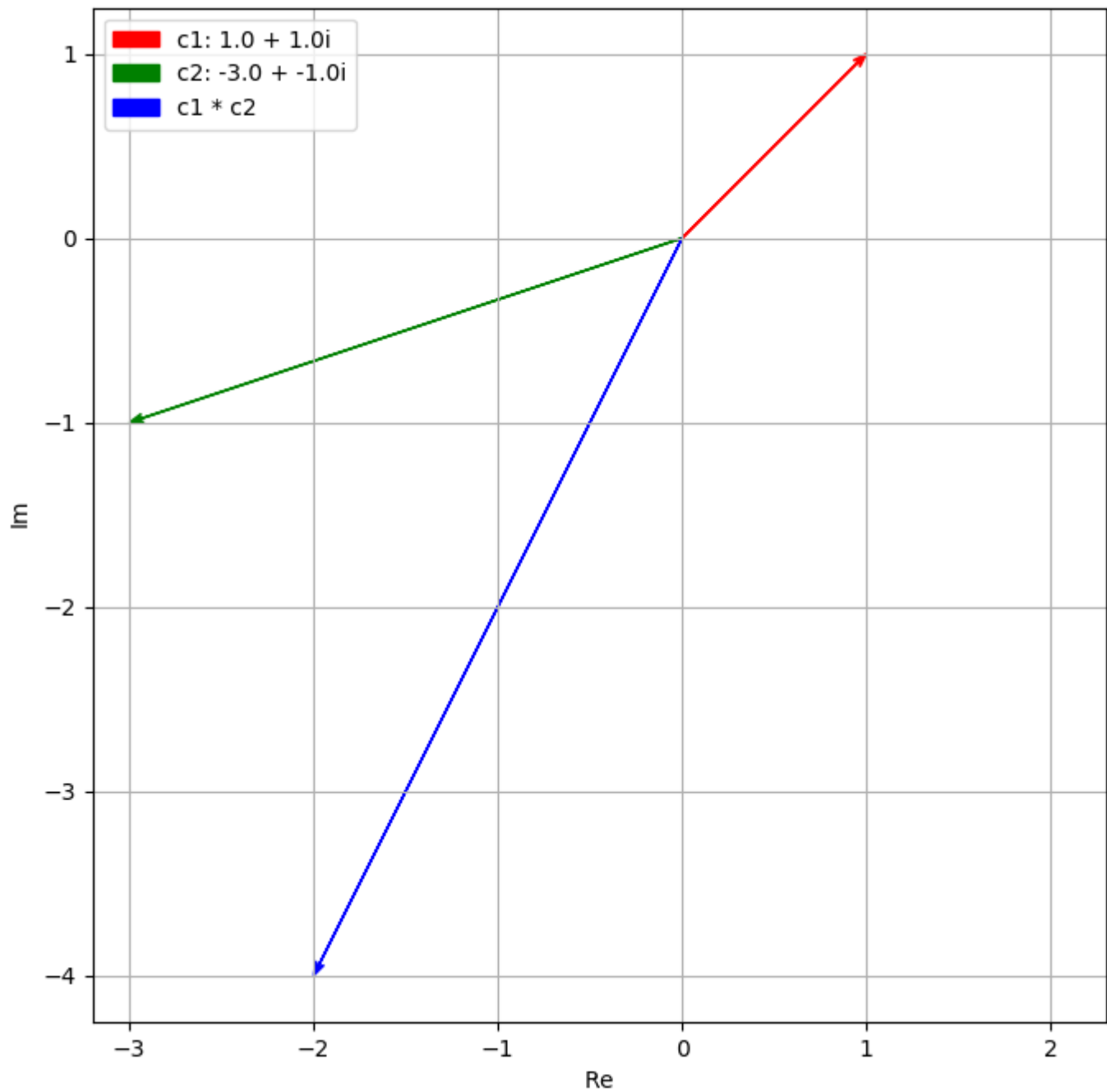










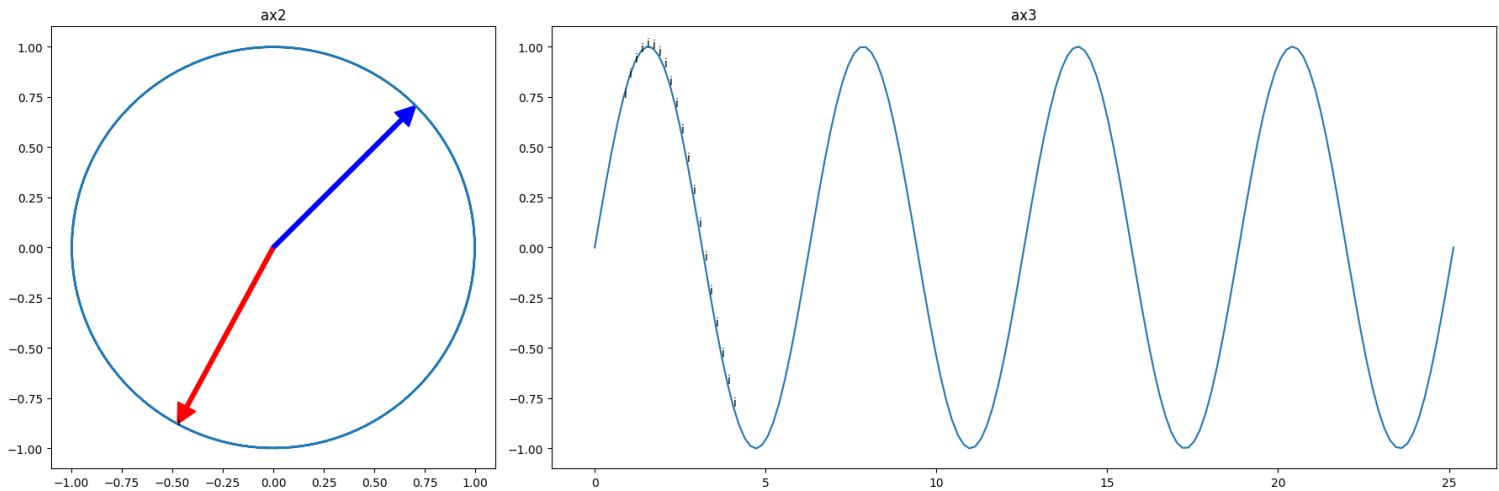


```
#import lib.main.e5_i_anim_hyperplot
print ("c3; red start and blue end")
import lib.main.c3_i_anim_circle_rotate_plot
import numpy as np
lib.main.c3_i_anim_circle_rotate_plot.plot_i_circle(np.pi/4)
```

```
c3; red start and blue end
lib.main.e5_i_anim_hyperplot
not via e5 main
/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno_book1/imgnobk1
not vai c3 main
```

```
index: 25
theta4 at 4.216902890724555
a= -0.4754593388242117 b= -0.8797376979104871
```

```
/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno_book1/imgnobk1/lib/ma
plt.tight_layout()
```



and if you change  $\theta$  you can see

- a one dimension being (on x) a periodic change (like as real number being we see  $i, -1, -$
- or for 2 dimension being (on  $i$  and  $x$ ), you can see a wave!!!

Any number on this 2 dimension plane can work like any other number It can be added, subtracted, multiplication and divide. (This is different from vector.)

Not just that, given we have a multiplication, and we know there is rotation involve, one can separate any complex number into another 2 dimension (not like  $x$  and  $y$ ), but its multiplication character and its rotation character.

Looking at real number  $c = 2 * 3$   $c$  will be double of 3, or if we  $*c$  it will be  $6*$  hence we know  $*z$  would have one dimension of multiplication of its magnitude, so ...

$$z = \text{magnitude of } z * (z / \text{magnitude of } z) \text{ or } |Z| * Z / |Z|$$

[Skip to main content](#)



i.e. we know that  $*z$  will mean using  $|Z|$  to multiply and  $Z/|Z|$  to rotate

In fact we call this the polar form i.e.  $z = |Z| * f(\theta)$  or  $A * f(\theta)$

--> and in the diagram we can even represent it as

$$z = A * (\cos\theta + i\sin\theta)$$

BTW to calculate  $A$  one easy way because complex number is 90 degree rotation, it follows Euclidian Geometry or

$$z = a + bi \text{ then}$$

$$|z| = \sqrt{a^2 + b^2}$$

$$= (a + bi)(a - bi)$$

where  $a - bi$  is the conjugate of  $z$

we will find this useful when the rotation is not Euclidian ...

```
print ("another example - c3; red start and blue end")
import lib.main.c3_i_anim_circle_rotate_plot
import numpy as np
#lib.main.c3_i_anim_circle_rotate_plot.do_anim()
# need to run in terminal for the moment???
```

another example - c3; red start and blue end

$$z = A * (\cos\theta + i\sin\theta)$$

This does not help us very much as a formula goes

If we rotate  $z_1$  and the  $z_2$

$z_1 * z_2$  involving cos and sin and is very painful

However, euler has studied and found out that

[Skip to main content](#)

$e^{i\theta} = \cos\theta + i\sin\theta$  ← the euler equation

substitutue it one can get the rotation part is actually  $e^{i\theta}$

or  $z = Ae^{i\theta}$

For example  $z_1 * z_2 = A_1 * A_2(e^{i\theta_1})(e^{i\theta_2}) = A_1 * A_2 * e^{i(\theta_1 + \theta_2)}$

← NO *cosine* and *sine* but multiplication, addition and power (another thankful one is  $(\cos\theta + i\sin\theta)^n = (\cos n\theta + i\sin n\theta)$  Moivre and Euler

This will help the next 100+ years of engineering, and maths and ... as rotation is everywhere and now adding of rotation is really just addition!

As beauty if one substitue  $\pi$  into that euler equation, we have what many call the most beautiful mathmathical equation :

$$e^{i\pi} = -1$$

and may we note:

$-1$  is arthmetic unusal thing

$\pi$  is geometry unusal thing

$i$  is algebera unusal thing

$e$  is calculus unusal thing

They are from total different field of human endeavour ... and somehow these unusal all join up into this

EULER IDENTITY

Unfortunately and may be fortunately someone observe that i nature is rotation and hence if one observe how i rotate, one can generate wave as seen above.

And if one note that the change of this i is related back to itself, a guy actually create a quantum mechanic out of this imaginary number

change of wave over *time* =  $i * [\dots]$  \* wave itself ← basic Schrödinger equation

[Skip to main content](#)

we can only observe real number The rotation is there but we still see the real number Like the one dimension man only see  $x$  not  $i$  in the rotation demo Why one can use magnitude to observe as probability Random is built-in and determine in the last instance and not before And what happen to the The Schrödinger Cat then

Discrete and minimum energy, ... any other discrete

Uncertainty principles ...

BTW, real world offer more rotation ...

not only we have rotate  $360$  as  $i$ , but we also rotate  $720$  degree to go back to itself ... and like the song "love always around us" in 4 wedding, it is all around us! may be in the future we talk about electron!

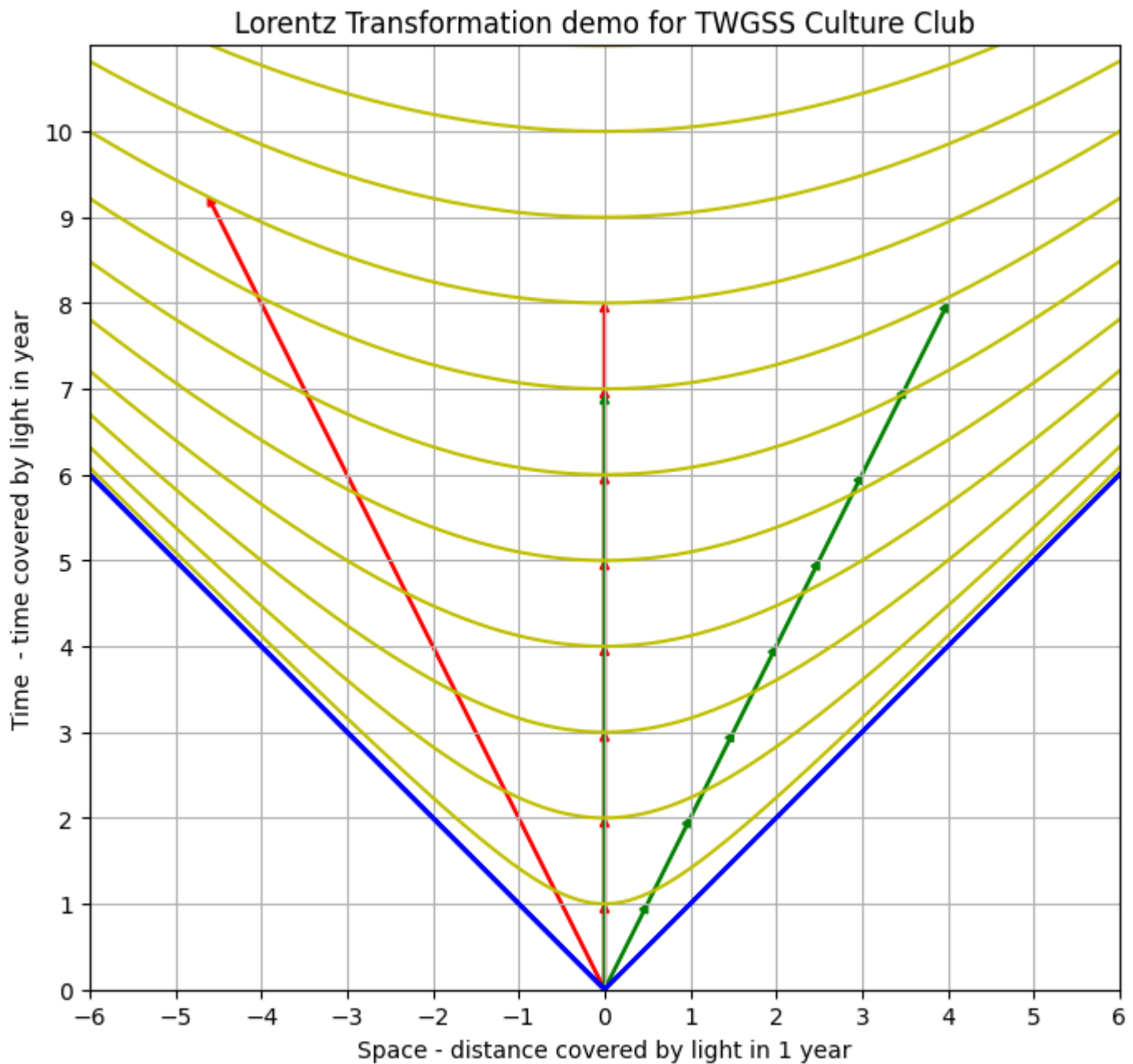
Also, you may note it is  $i$ , but actually someone has expanded into 4 dimension and it is use in every computer game when you rotate a character you fit the transformation into this 4 dimension rotation ...

## 4. Imagine another imaginery number and split it out

```
import lib.main.e5_i_anim_hyperplot

print(lib.main.e5_i_anim_hyperplot.lorentz_t_t00())
print(lib.main.e5_i_anim_hyperplot.lorentz_t_f00())
lib.main.e5_i_anim_hyperplot.hyperplot_and_tell()
```

```
lib.main.e5_i_anim_hyperplot
not via e5 main
[0, 6.928203230275509]
[-4.618802153517006, 9.237604307034012]
```



A taste of part 2

- if quantum mechanics is about the rotation, one observation is that  $i$  does not change during the rotation
- the reason is that its magnitude is defined for  $a + bi \rightarrow (a + bi)(a - bi)$  or  $a^2 + b^2$

What if we invent another imaginary number called split complex number where

$$i^2 = +1$$

[Skip to main content](#)

The issue is that what this imaginary number rotate. The magnitude should be constant

$A = (a + bj)(a - bj) = a^2 - b^2$ . Note the -ve sign there.

It is a hyperbola not a circle it will be draw when it rotate.

Use?

Well, it turns out our spacetime are related like this

$(ct)^2 - x^2 - y^2 - z^2$  or if concentrate on only time and say x dimension movement it  
 $a^2 - b^2$  format

all those talk about time dilution and length contraction ... all because of this split

Not only we live in imaginary number as a quantum being

We also live in split imaginary number as a relativity being

I am not sure I want to got the third imaginary number well ...

Part 2 (to be developed)

—> Draft to be tidy up with demo

– what if our world use this split complex number as above using  $j^2 = 1$  (instead of  $i^2 = -1$ )

Unlike Galileo and Newton which use absolute time and space and hence change observation frame by traveling say on a train does not affect the time and space. Your worldline shift (and not rotate) ... but this is against the issue that one thing does not shift or light speed cannot be changed.

Here the invariant is not time or space but a join number by them (or split complex number).

More importantly the light wave has zero split complex number and hence whatever you “rotate” its speed does not affect because it is 0.  $0 \times \text{whatsoever}$  is still 0.

Or  $0 = x + tj$  (here  $j$  is in time and  $s$  in distance, with  $x^2 - t^2 = \text{constant}$ )

In fact solve this equation  $x = \pm t$  and by using a proper light always travel at 45 degree (or  $\pi/4$  radians) and 135 degree ... i.e. just  $x = t$  and  $x = -t$  (scale -  $t$  in year say and  $x$  is distance travel by light in 1 year)

— — — — —

[Skip to main content](#)

To understand

Yes it helps a bit in understand one of the strange thing about relativity. The key issue is about the Galilei relativity (the shift not rotate of timeline)

Graphic

and

The strange way our real world operate or there is a constant that is independent from our frame of reference (at 0 or constant speed). 0 set the limit and non-0 confine us to  $< c$ .

—

Graphic see

One observer (red, point upward as  $x=0$  and  $t$  ever-increasing) Two observer (red + green) View from first (red point upward and green to the right) View from second (red point to the left and green now point upwards as at rest) Note the rotation is using hyperbola as whatever frame rotate there is a constant  $s^2 = t^2 - x^2$

And most important if you just look at time, you can see it shorten and so is length. Because it is the difference between time and space that matters. Individual dimension it change this way because of the negative side. (BTW, some textbook do  $x^2 - t^2$ .)

Part 2 end

1. Twin Paradox (a third blue observer, acceleration or doppler)
2. Geometry change from inertia to acceleration or Rindler not Gravity
3. Then how about the whole geometry from flat to curved due to mass (General, using projective and differential geometry, plus group ...) (Not about acceleration as in 2, as gravity is not acceleration)
4. other imagery or complex number like dual number

<— Draft to be tidy up with demo

Finally, the complex number and split complex number has joined as Quantum Field Theory  
Or the wave travel in relativity manner,  
— its vibration follow the complex number

[Skip to main content](#)

Unfortunately we still do not have the wave onto the general relativity  
-- projective geometry and differential geometry ...

Thanks for your patient and attention.

We live briefly in these 2 imaginary number (and more) world.

What you see is NOT real or at least NOT JUST REAL!

## Appendix - may be needed for reference

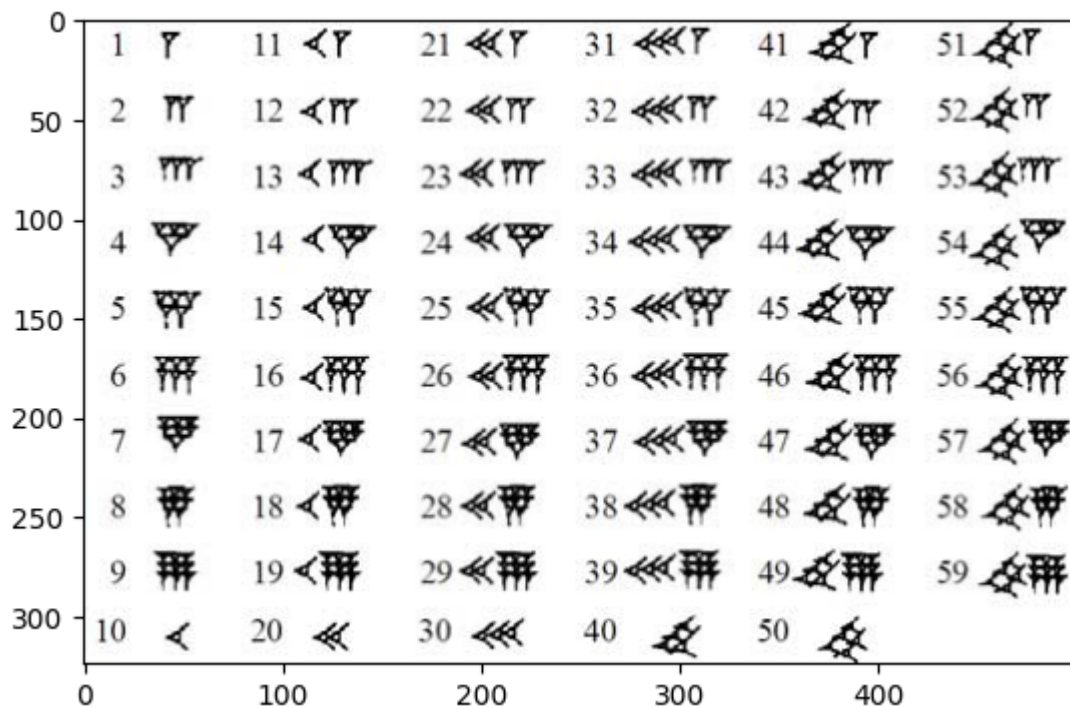
```
import os  
print(os.getcwd())
```

```
/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno_book1/imgnobk1
```

```
import lib.main.a0_babylon_pos_0
```

```
not in main of a0
```

```
lib.main.a0_babylon_pos_0.display_img(1)
```



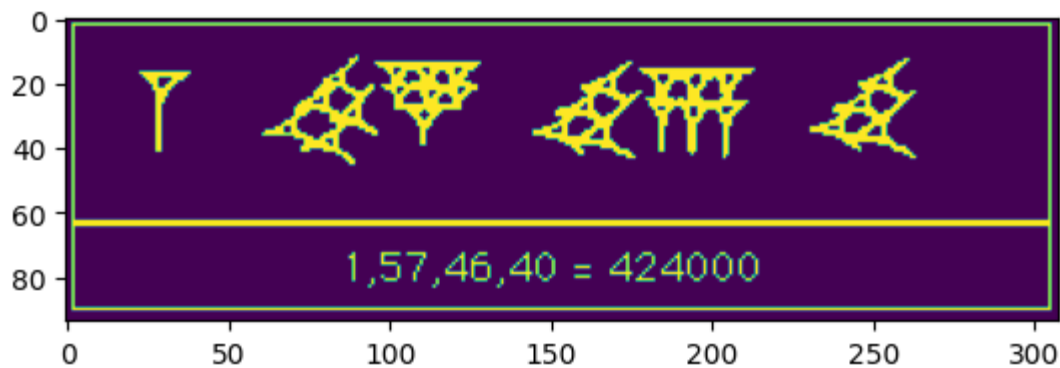
note

a) there are only 2 symbols 1 and 10

b) every number is actually by placing and counting number 1 and number 10 out;

c) reaching 59 then what ... 60 is a problem let us skip it first ;-P

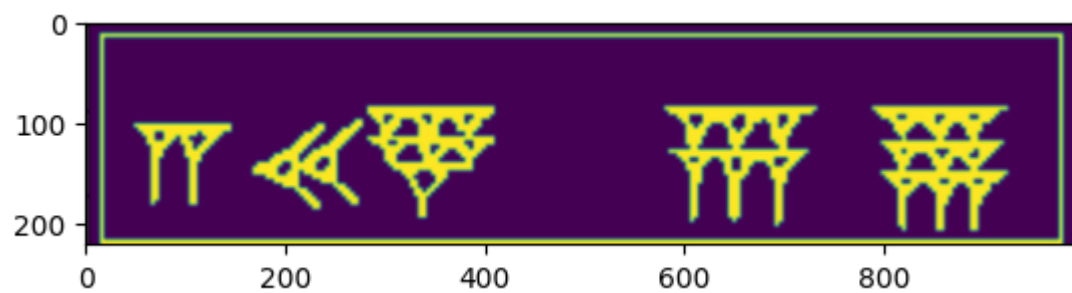
```
lib.main.a0_babylon_pos_0.display_img(2)
```



```
lib.main.a0_babylon_pos_0.display_img(3)
```

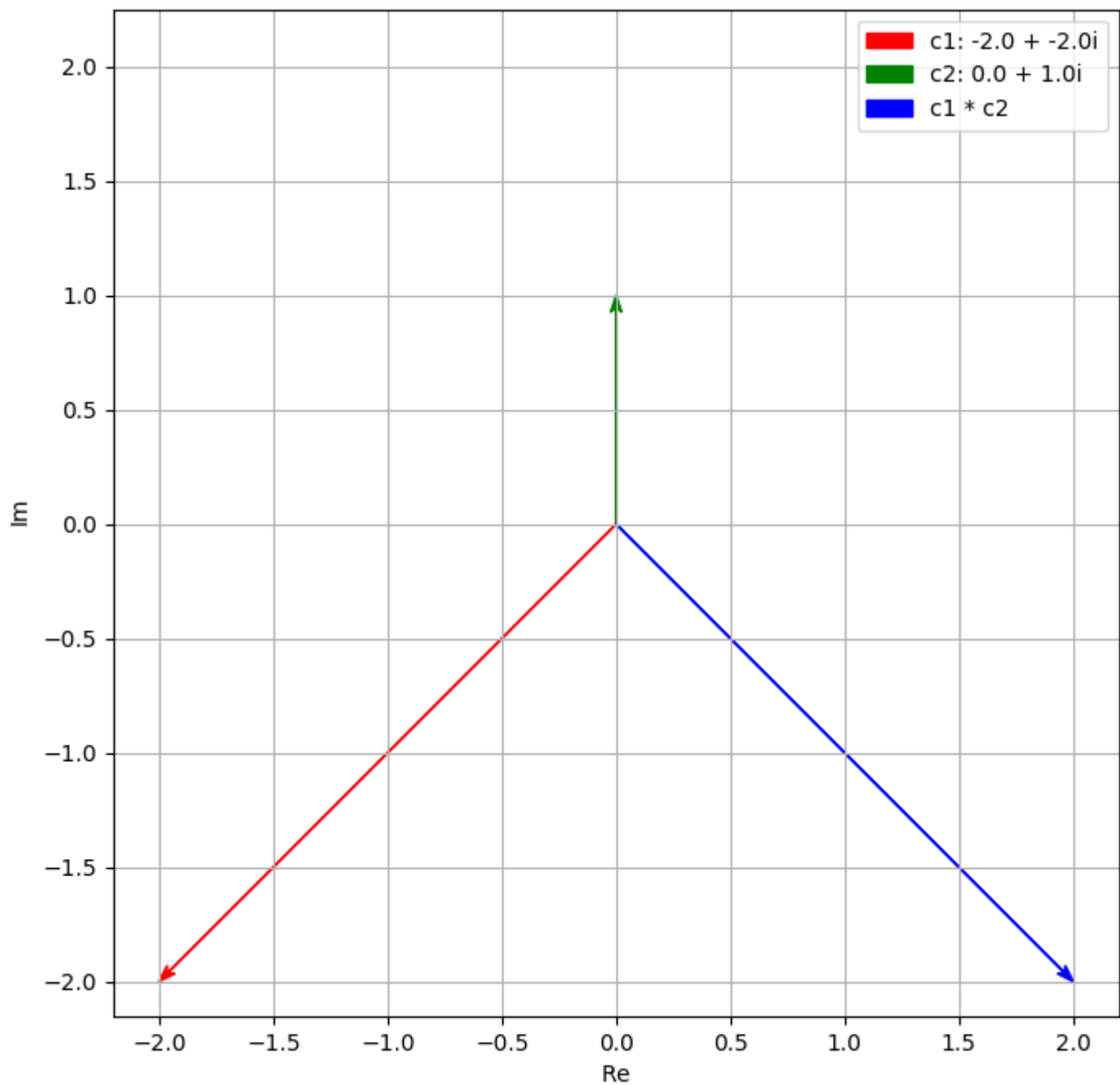
[Skip to main content](#)





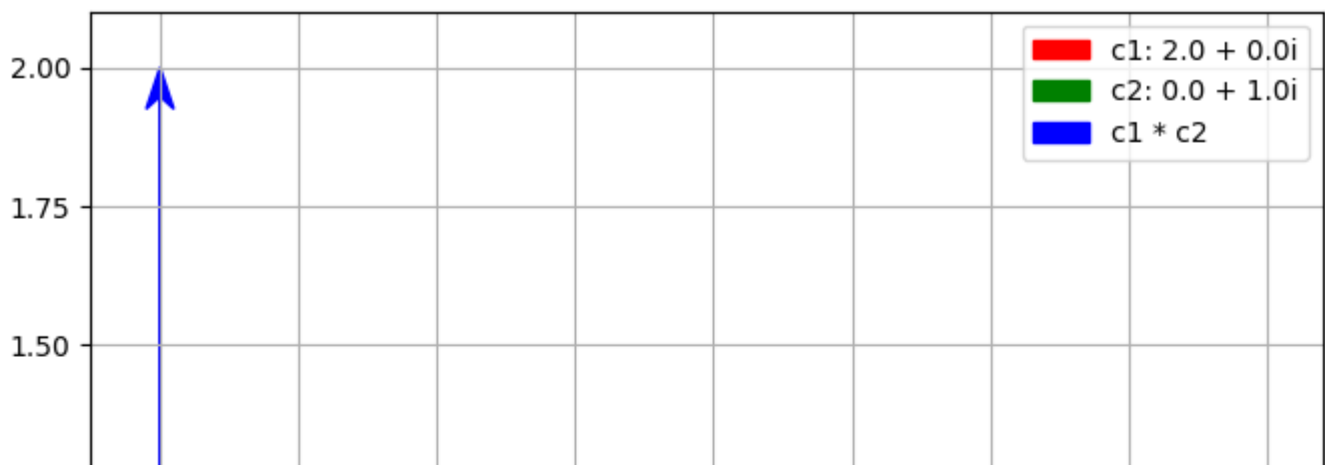
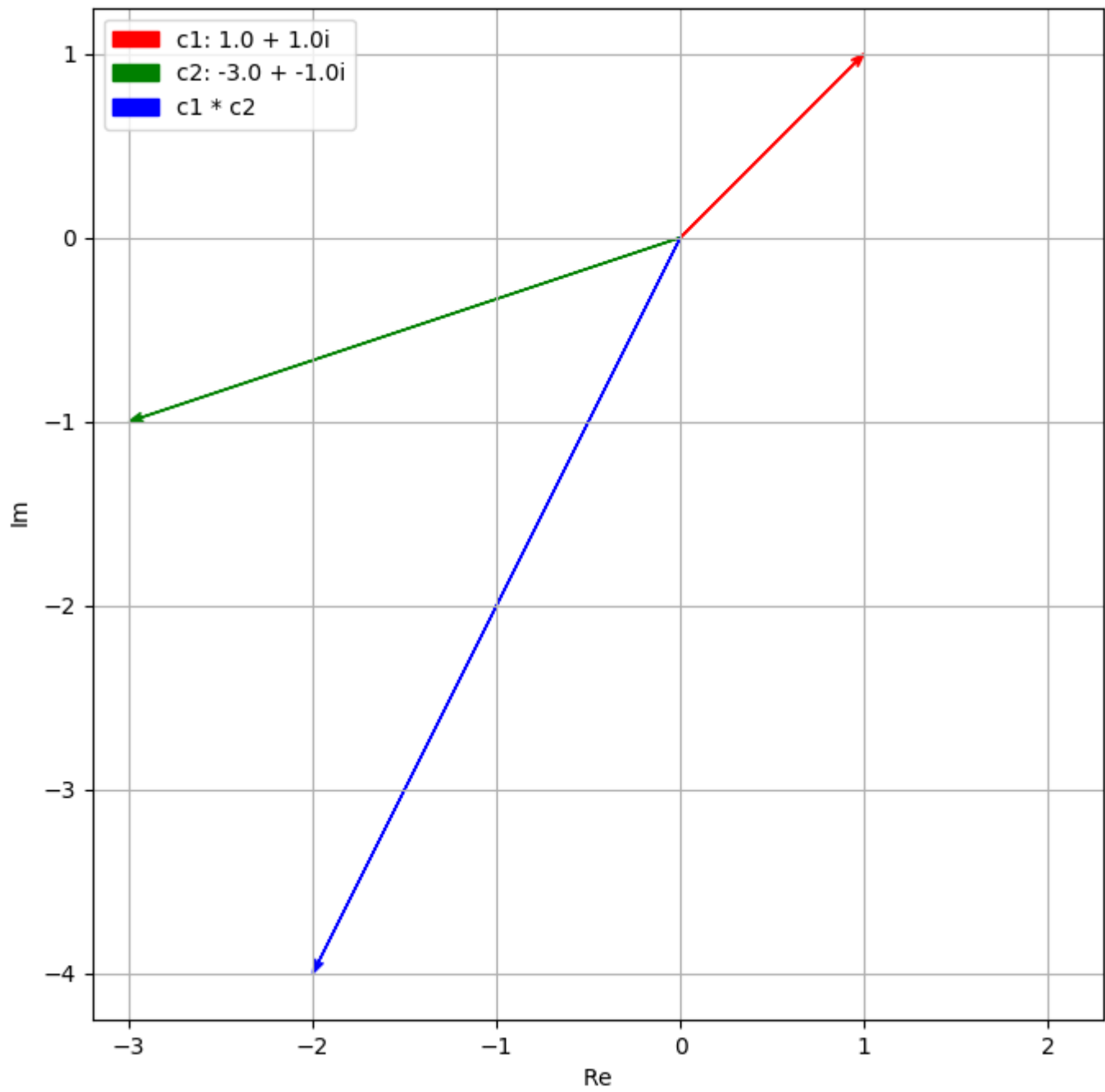
```
%matplotlib inline
import lib.main.a1_i_mul_plot
```

lib.main.a1\_i\_mul\_plot  
not via main  
it is a rotation  
and i always rotate 90 degree or  $\pi/2$  radians

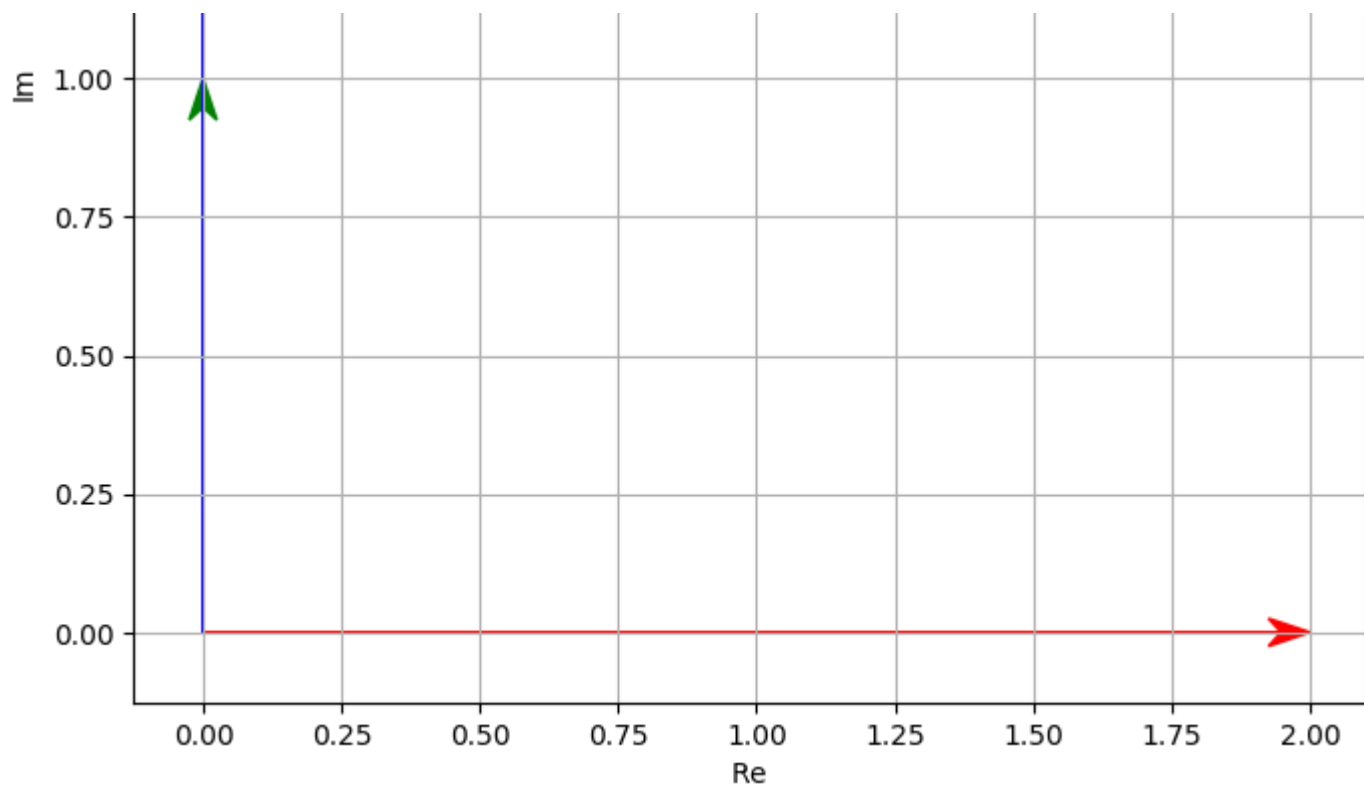


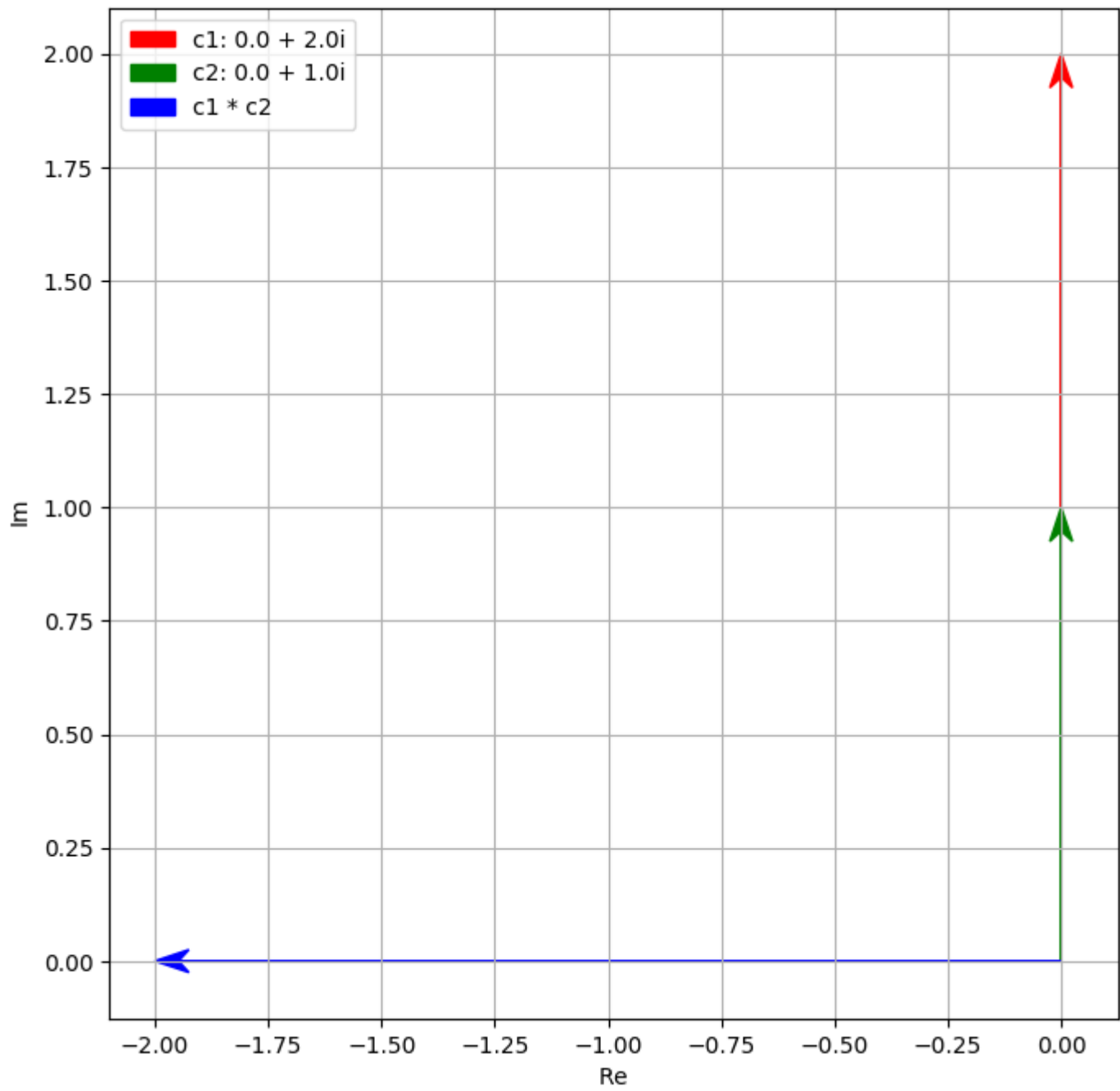
```
%matplotlib inline  
lib.main.a1_i_mul_plot.some_plot()
```

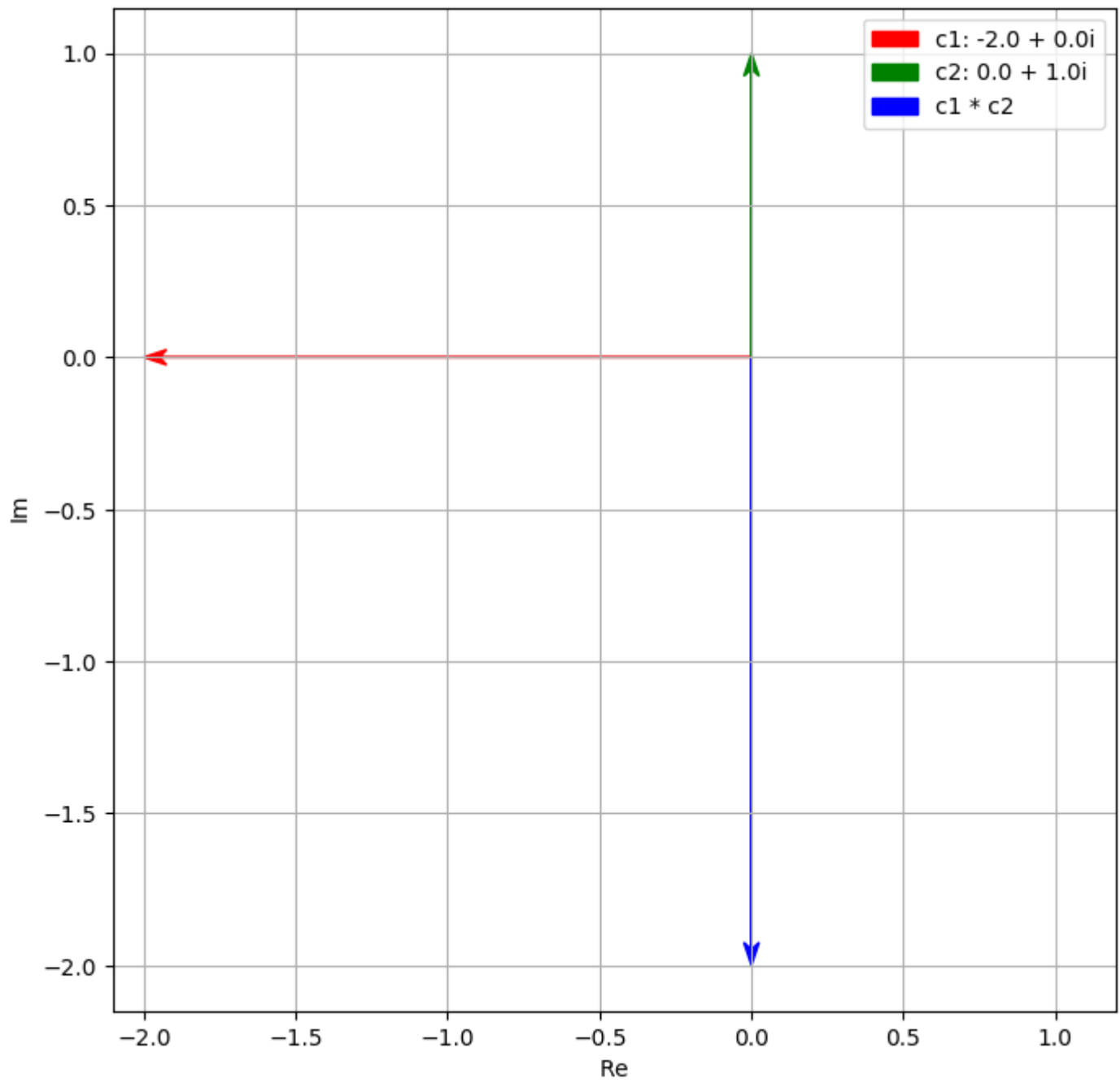
-- more example--

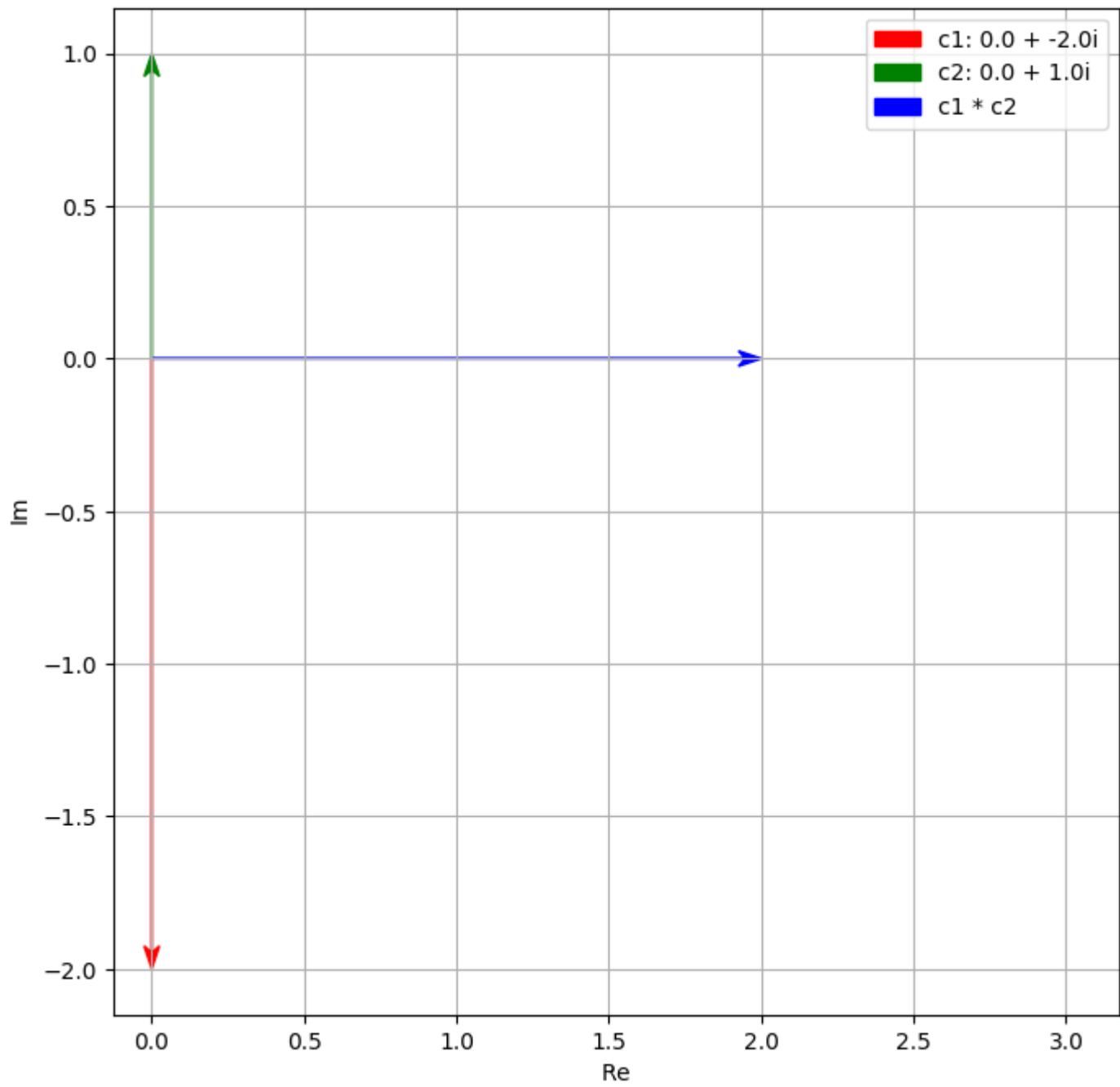


[Skip to main content](#)

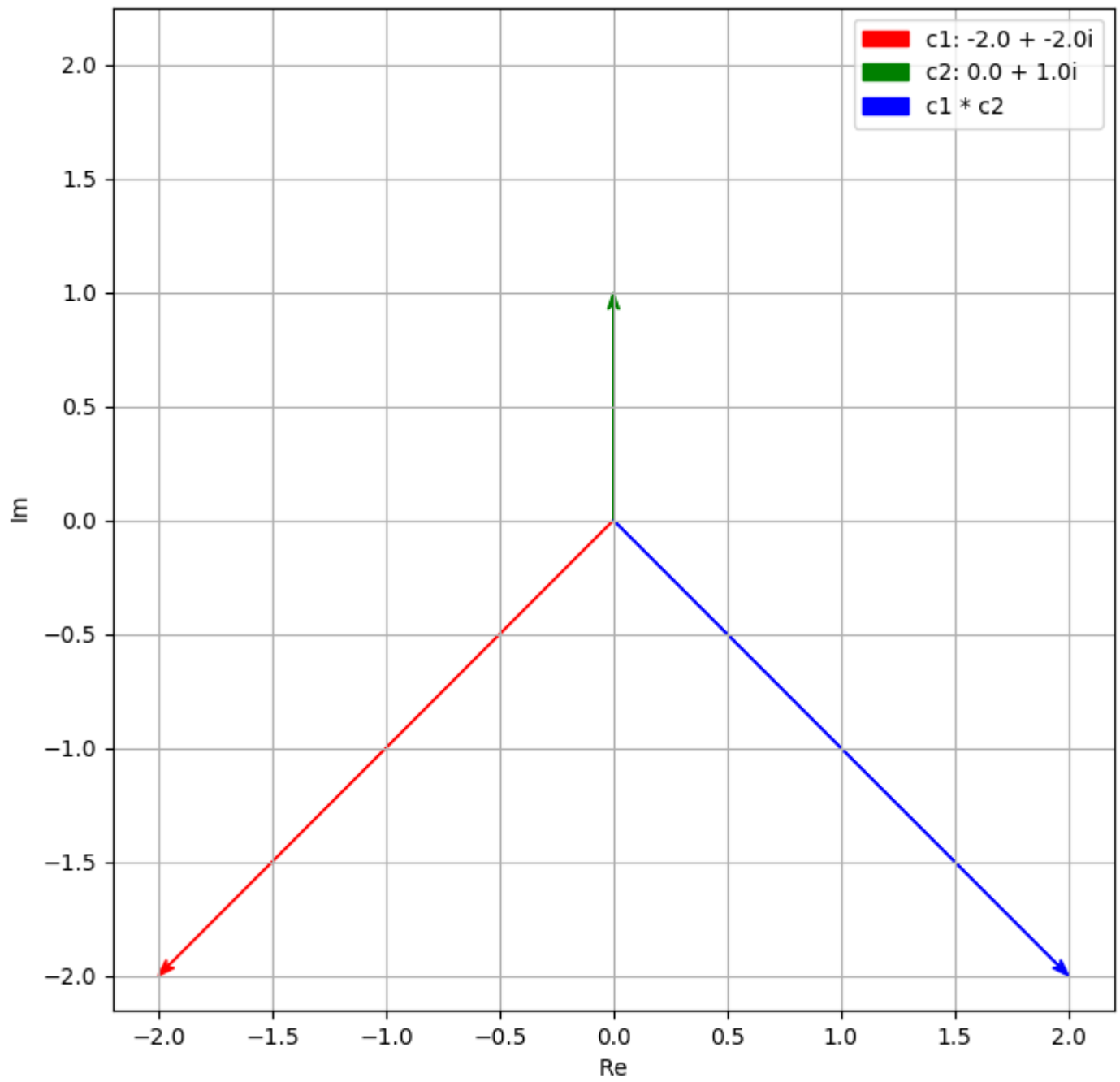


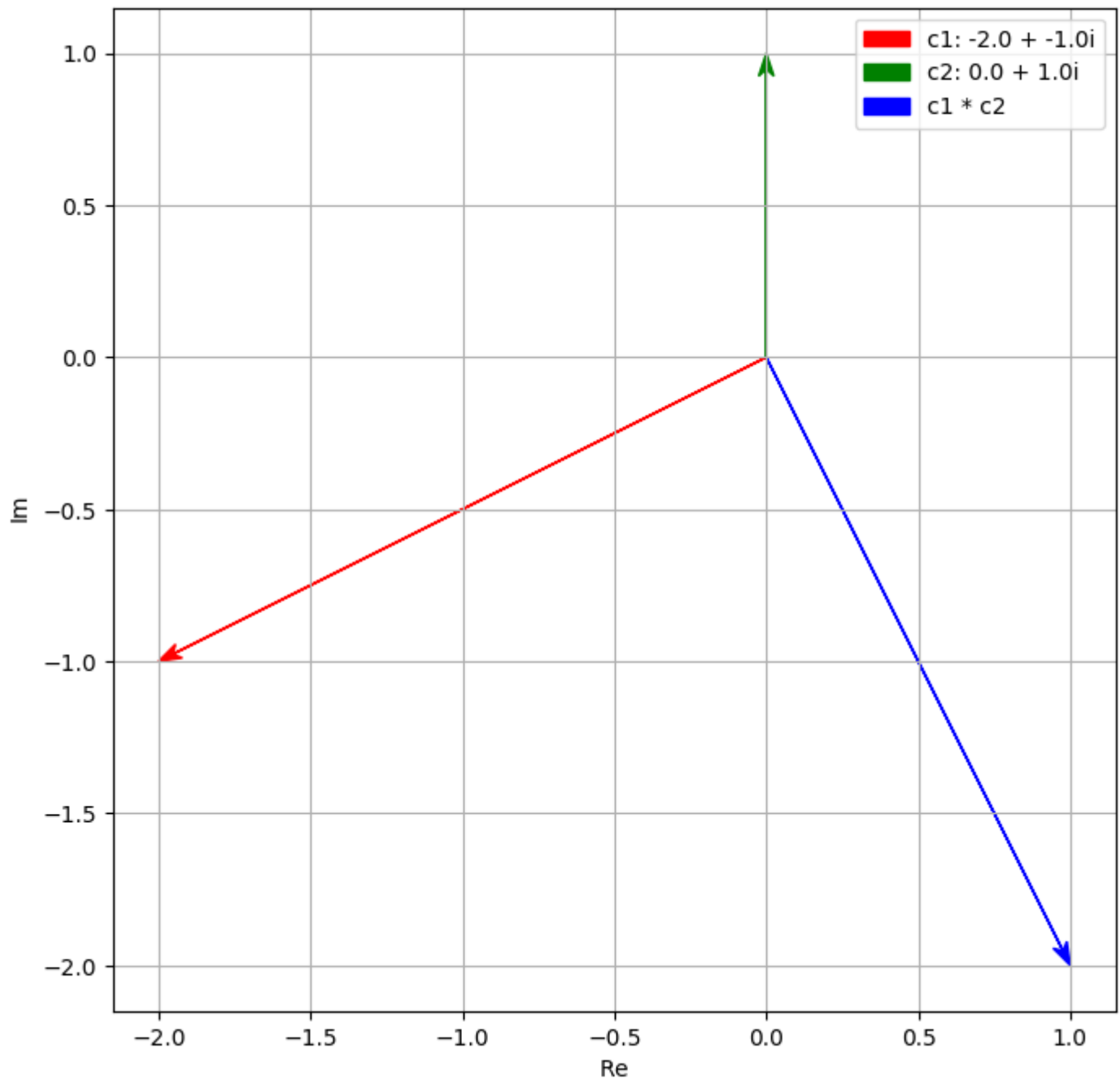


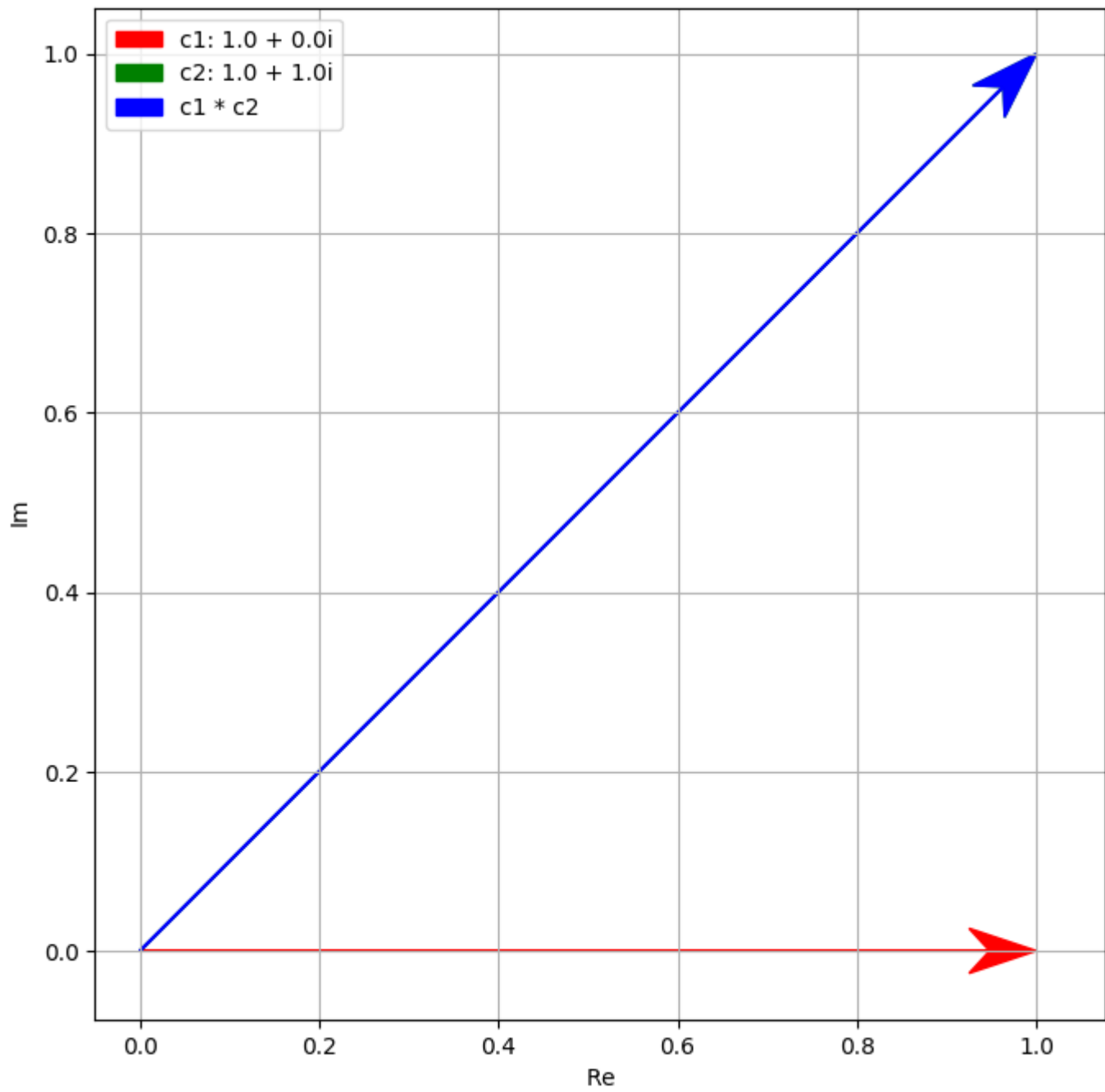


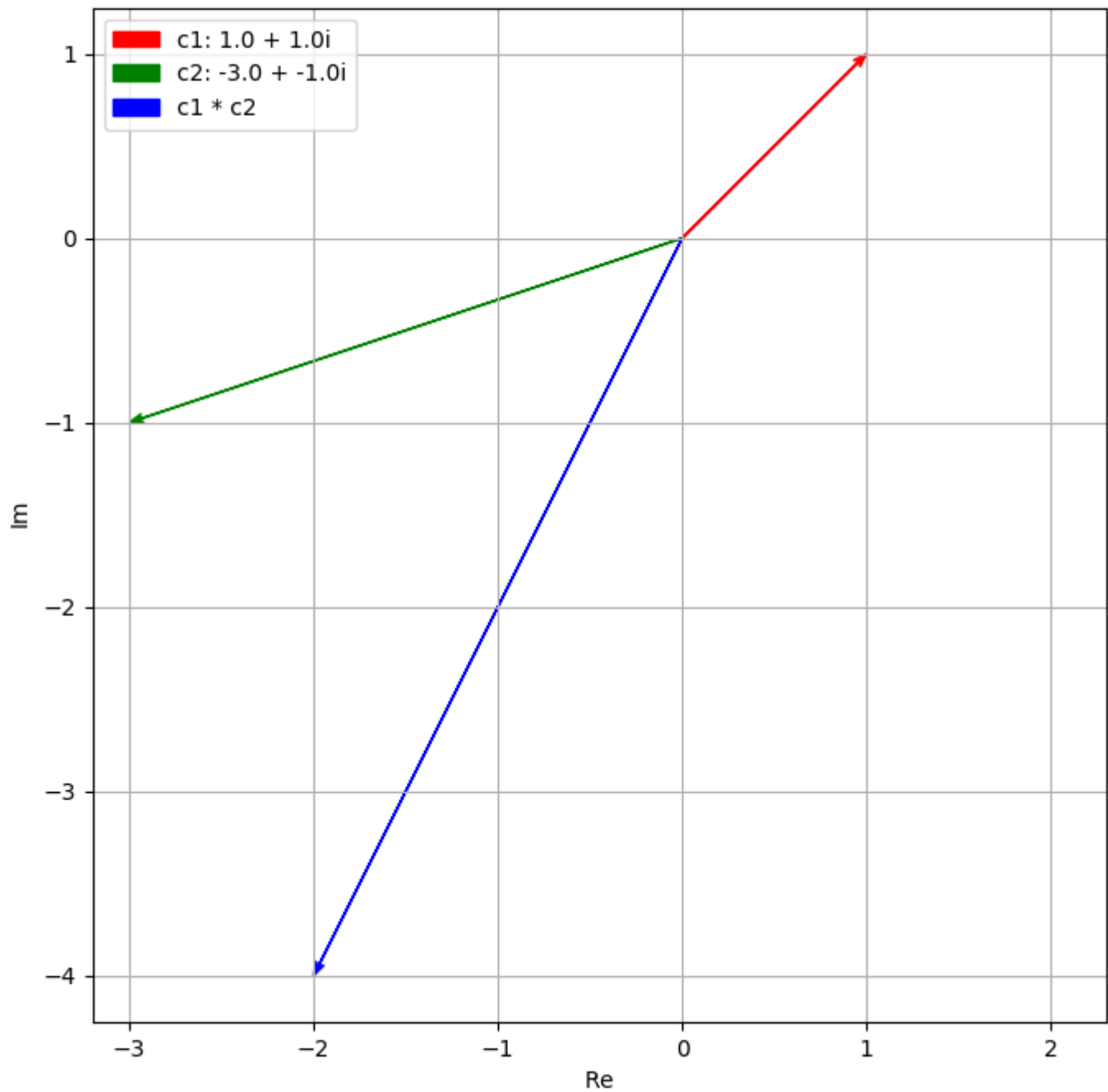










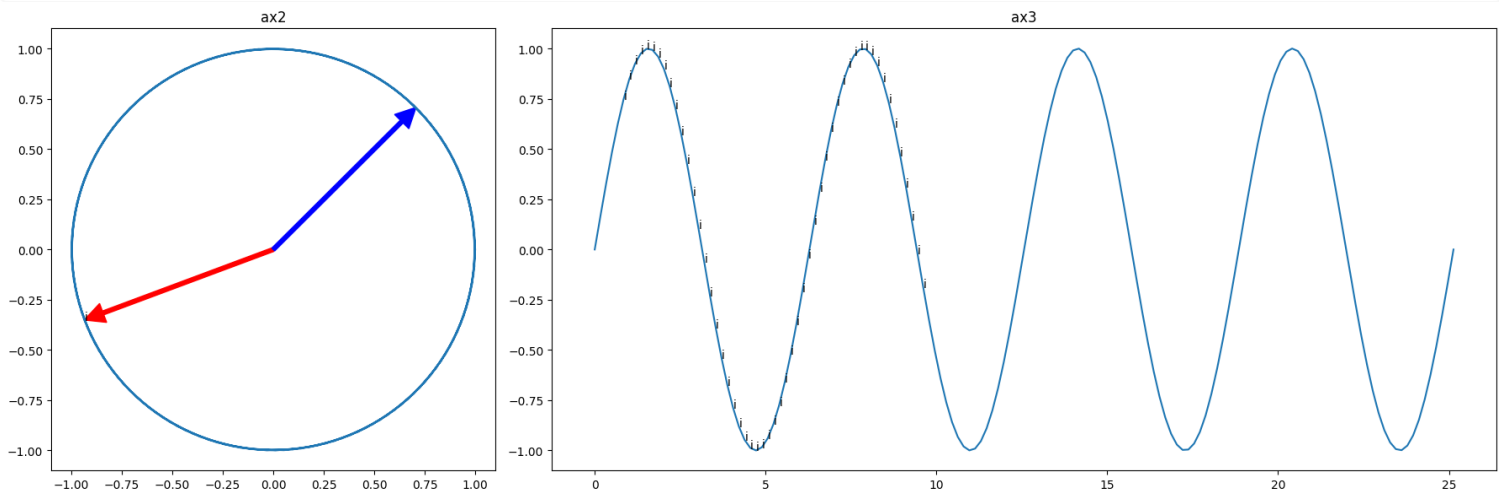


```
#import lib.main.e5_i_anim_hyperplot
print ("c3; red start and blue end")
import lib.main.c3_i_anim_circle_rotate_plot
import numpy as np
lib.main.c3_i_anim_circle_rotate_plot.plot_i_circle(np.pi/4)
```

```
c3; red start and blue end
lib.main.e5_i_anim_hyperplot
not via e5 main
/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno_book1/imgnobk1
not vai c3 main
```

```
index: 58
theta4 at 9.783214706480967
a= -0.9364463741042691 b= -0.3508107587104012
```

```
/Users/ngcchk/Documents/Github/gpd2-win-unity1/ipadred-rain/imgno_book1/imgnobk1/lib/ma
plt.tight_layout()
```



```
print ("another example - c3; red start and blue end")
import lib.main.c3_i_anim_circle_rotate_plot
import numpy as np
#lib.main.c3_i_anim_circle_rotate_plot.do_anim()
# need to run in terminal for the moment???
```

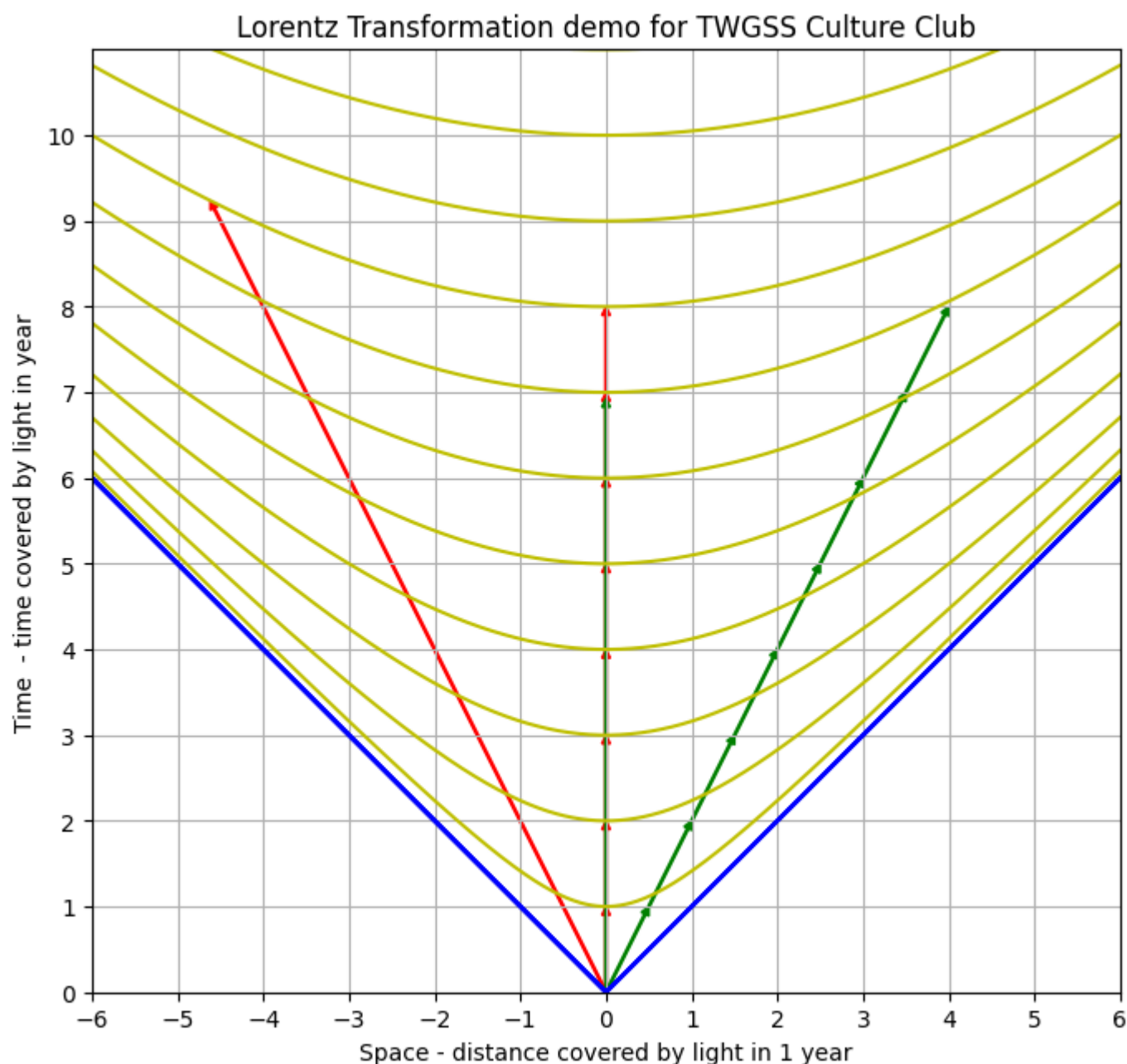
another example - c3; red start and blue end

```
import lib.main.e5_i_anim_hyperplot

print(lib.main.e5_i_anim_hyperplot.lorentz_t_t00())
print(lib.main.e5_i_anim_hyperplot.lorentz_t_f00())
lib.main.e5_i_anim_hyperplot.hyperplot_and_tell()
```

[Skip to main content](#)

```
[0, 6.928203230275509]  
[-4.618802153517006, 9.237604307034012]
```



## Welcome to your Jupyter Book

This is a small sample book to give you a feel for how book content is structured. It shows off a few of the major file types, as well as some sample content. It does not go in-depth into any particular topic - check out [the Jupyter Book documentation](#) for more information. See also [Jupyter installtion](#)

[Skip to main content](#)

# Page title

[Section 1 \(will be listed\)](#).

- [Sub-section 1 \(will be listed\)](#).

[Section 2 \(will be listed\)](#).

## [Section 1 \(will be listed\)](#).

### [Sub-section 1 \(will be listed\)](#).

## [Section 2 \(will be listed\)](#).

# Markdown Files

Whether you write your book’s content in Jupyter Notebooks (`.ipynb`) or in regular markdown files (`.md`), you’ll write in the same flavor of markdown called **MyST Markdown**. This is a simple file to help you get started and show off some syntax.

## What is MyST?

MyST stands for “Markedly Structured Text”. It is a slight variation on a flavor of markdown called “CommonMark” markdown, with small syntax extensions to allow you to write **roles** and **directives** in the Sphinx ecosystem.

For more about MyST, see [the MyST Markdown Overview](#).

## Sample Roles and Directives

Roles and directives are two of the most powerful tools in Jupyter Book. They are kind of like functions, but written in a markup language. They both serve a similar purpose, but roles are written in one line

[Skip to main content](#)

whereas **directives span many lines**. They both accept different kinds of inputs, and what they do with those inputs depends on the specific role or directive that is being called.

Here is a “note” directive:

#### Note

Here is a note

It will be rendered in a special box when you build your book.

Here is an inline directive to refer to a document: [Notebooks with MyST Markdown](#).

## Citations

You can also cite references that are stored in a `bibtex` file. For example, the following syntax:

`{cite}`holdgraf_evidence_2014`` will render like this: [\[HdHPK14\]](#).

Moreover, you can insert a bibliography into your page with this syntax: The `{bibliography}` directive must be used for all the `{cite}` roles to render properly. For example, if the references for your book are stored in `references.bib`, then the bibliography is inserted with:

[\[HdHPK14\]](#) Christopher Ramsay Holdgraf, Wendy de Heer, Brian N. Pasley, and Robert T. Knight. Evidence for Predictive Coding in Human Auditory Cortex. In *International Conference on Cognitive Neuroscience*. Brisbane, Australia, Australia, 2014. Frontiers in Neuroscience.

## Learn more

This is just a simple starter to get you started. You can learn a lot more at [jupyterbook.org](https://jupyterbook.org).

## Content with notebooks

You can also create content with Jupyter Notebooks. This means that you can include code blocks and their outputs in your book.

[Skip to main content](#)



# Markdown + notebooks

As it is markdown, you can embed images, HTML, etc into your posts!



## Markedly Structured Text

You can also *add<sub>math</sub>* and

*math<sup>blocks</sup>*

or

*mean<sub>la<sub>tex</sub></sub>*

*mathblocks*

But make sure you  $\$$ Escape  $\$$ your  $\$$ dollar signs  $\$$ you want to keep!

## MyST markdown

MyST markdown works in Jupyter Notebooks as well. For more information about MyST markdown, check out [the MyST guide in Jupyter Book](#), or see [the MyST markdown documentation](#).

[Skip to main content](#)

# Code blocks and outputs

Jupyter Book will also embed your code blocks and output in your book. For example, here's some sample Matplotlib code:

```
from matplotlib import rcParams,ycler
import matplotlib.pyplot as plt
import numpy as np
plt.ion()
```

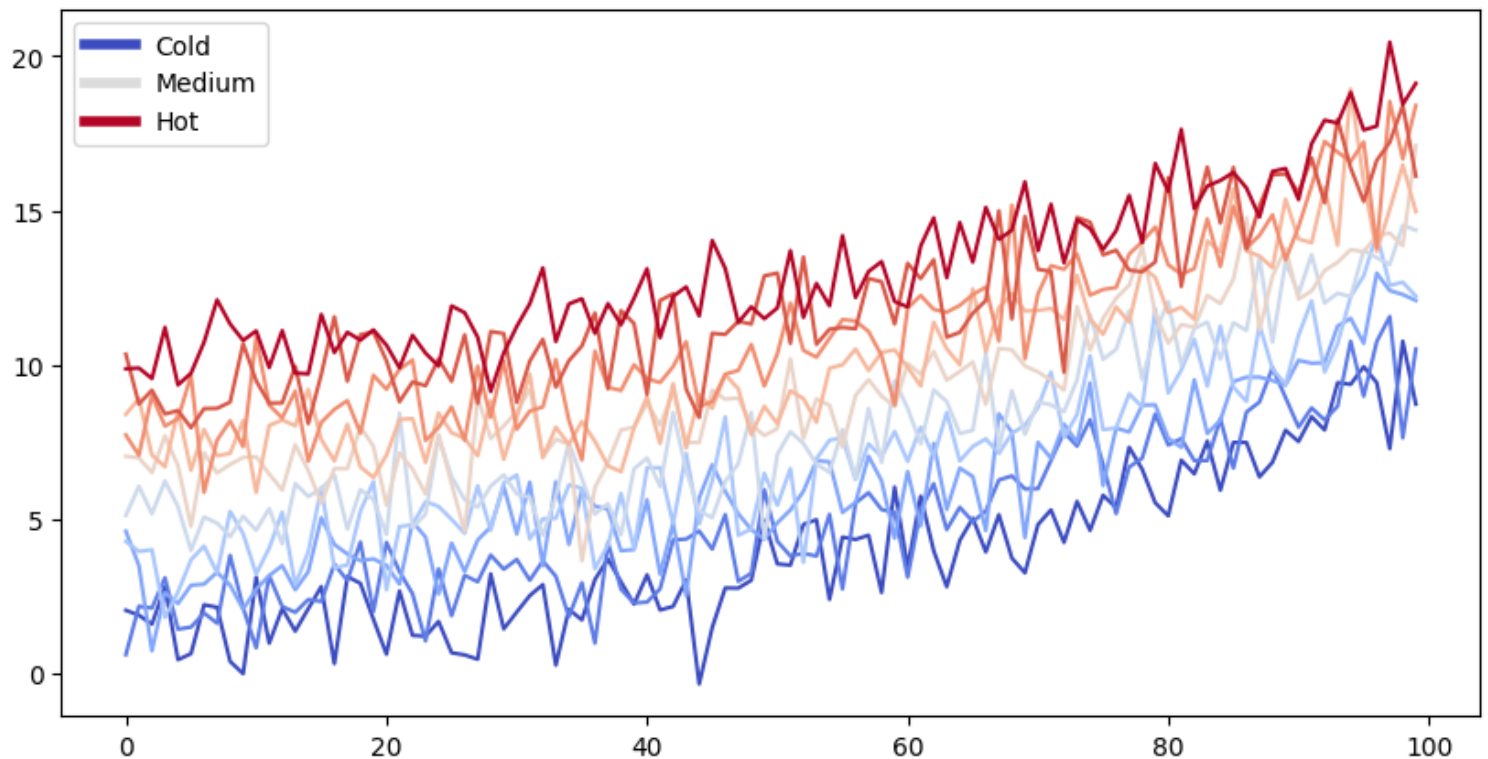
```
<contextlib.ExitStack at 0x1101af190>
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)

N = 10
data = [np.logspace(0, 1, 100) + np.random.randn(100) + ii for ii in range(N)]
data = np.array(data).T
cmap = plt.cm.coolwarm
rcParams['axes.prop_cycle'] = cycler(color=cmap(np.linspace(0, 1, N)))

from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color=cmap(0.), lw=4),
                 Line2D([0], [0], color=cmap(.5), lw=4),
                 Line2D([0], [0], color=cmap(1.), lw=4)]

fig, ax = plt.subplots(figsize=(10, 5))
lines = ax.plot(data)
ax.legend(custom_lines, ['Cold', 'Medium', 'Hot']);
```



There is a lot more that you can do with outputs (such as including interactive outputs) with your book. For more information about this, see [the Jupyter Book documentation](#)

## Notebooks with MyST Markdown

Jupyter Book also lets you write text-based notebooks using MyST Markdown. See [the Notebooks with MyST Markdown documentation](#) for more detailed instructions. This page shows off a notebook written in MyST Markdown.

### An example cell

With MyST Markdown, you can define code cells with a directive like so:

```
print(2 + 2)
```

4

When you build the contents of your book, the code cells will be executed with your default

[Skip to main content](#)

## ➔ See also

Jupyter Book uses [Jupyter](#) to convert text-based files to notebooks, and can support [many other text-based notebook files](#).

# Create a notebook with MyST Markdown

MyST Markdown notebooks are defined by two things:

1. YAML metadata that is needed to understand if / how it should convert text files to notebooks (including information about the kernel needed). See the YAML at the top of this page for example.
2. The presence of `{code-cell}` directives, which will be executed with your book.

That's all that is needed to get started!

## Quickly add YAML metadata for MyST Notebooks

If you have a markdown file and you'd like to quickly add YAML metadata to it, so that Jupyter Book will treat it as a MyST Markdown Notebook, run the following command:

```
jupyter-book myst init path/to/markdownfile.md
```

# testing jupyter-book failed using fAe examples

from [markjay4k/fourier-transform](#) unless matplotlib notebook to inline

Not work immeidately

as jupyterlab assume using conda and use javascript for interaction ... that is not in the past

[jupyterlab/jupyterlab#3934](#)

as detailed discussed here: <https://stackoverflow.com/questions/51922480/javascript-error-ipython-is-not-defined-in-jupyterlab/56416229#56416229>

[Skip to main content](#)

```

import numpy as np

def rect(x, B):
    """
    create a rectangle function
    returns a numpy array that is 1 if  $|x| < w$  and 0 if  $|x| > w$ 
    B is the rectangle width centered at 0
    x is the number of points in the array
    """

    B = int(B)
    x = int(x)

    high = np.ones(B)
    low1 = np.zeros(int(x/2 - B/2))
    x1 = np.append(low1, high)
    rect = np.append(x1, low1)

    if x > len(rect):
        rect = np.append(rect, 0)
    elif x < len(rect):
        rect = rect[:-1]

    return rect

```

```

import numpy as np

def rect(x, B):
    """
    create a rectangle function
    returns a numpy array that is 1 if  $|x| < w$  and 0 if  $|x| > w$ 
    B is the rectangle width centered at 0
    x is the number of points in the array
    """

    B = int(B)
    x = int(x)

    high = np.ones(B)
    low1 = np.zeros(int(x/2 - B/2))
    x1 = np.append(low1, high)
    rect = np.append(x1, low1)

    if x > len(rect):
        rect = np.append(rect, 0)
    elif x < len(rect):
        rect = rect[:-1]

    return rect

```

```

%matplotlib notebook
# %matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# constants and x array
pi = np.pi
length = 2000
x = np.linspace(-1, 1, length)

# create figure and axes
fig, (ax1, ax2) = plt.subplots(2, figsize=(12, 6))

# creating our line objects for the plots
sinc, = ax1.plot(x, np.sin(x), '-b')
box, = ax2.plot(x, np.sin(x), '-r')

def animate(B):
    """
    this function gets called by FuncAnimation
    each time called, it will replot with a different width "B"

    B: rect width

    return:
        sinc: ydata
        box: ydata
    """

    # create our rect object
    f = rect(len(x), B)
    box.set_ydata(f)

    # create our sinc object
    F = (B / length) * np.sin(x * B / 2) / (x * B / 2)
    sinc.set_ydata(F)

    # adjust the sinc plot height in a loop
    ax1.set_ylim(np.min(F), np.max(F))

    # format the ax1 yticks
    plt.setp(ax1, xticks=[-0.25, 0.25], xticklabels=['-1/4', '1/4'],
             yticks=[0, np.max(F)], yticklabels=['0', 'B={:.2f}'.format((B / length))])

    # format the ax2 xticks to move with the box
    plt.setp(ax2, yticks=[0, 1],
             xticks=[-1, -1 * B / length, 1 * B / length, 1], xticklabels=['-1', '-B/2'])

def init():
    """
    initialize the figure
    """

```

[Skip to main content](#)

```

ax2.set_xlim(-1, 1)
ax1.axhline(0, color='black', lw=1)
ax2.axhline(0, color='black', lw=1)
plt.rcParams.update({'font.size':14})

return sinc, box,

```

```

# the FuncAnimation function iterates through our animate function using the steps array
step = 10
steps = np.append(np.arange(10, 1000, step), np.arange(1000, 10, -1 * step))
ani = FuncAnimation(fig, animate, steps, init_func=init, interval=50, blit=True)
plt.show()

```

```

import numpy as np

def rect(x, B):
    """
    create a rectangle function
    returns a numpy array that is 1 if |x| < w and 0 if |x| > w
    B is the rectangle width centered at 0
    x is the number of points in the array
    """

    B = int(B)
    x = int(x)

    high = np.ones(B)
    low1 = np.zeros(int(x/2 - B/2))
    x1 = np.append(low1, high)
    rect = np.append(x1, low1)

    if x > len(rect):
        rect = np.append(rect, 0)
    elif x < len(rect):
        rect = rect[:x]

    return rect

```

## testing jupyter-book ok using fFy examples

from [markjay4k/fourier-transform](#) as only use matplotlib inline

But there is no plot just wedge ...

1. Fourier Transform is a generalized version of the Fourier Series
2. It applies to both period and non periodic functions
  - For periodic functions, the spectrum is discrete
  - For non-period functions, the spectrum is continuous

## Definitions

### Fourier Transform

Fourier Transform of  $f(x)$  is  $F(k)$   $F(k) = \mathcal{FT}\{f(x)\}$

$$F(k) = \int_{-\infty}^{\infty} f(x) \exp(-ikx) dx$$

where  $k = \frac{2\pi}{x}$  is called the "wavenumber"

### Inverse Fourier Transform

To go back to  $f(x)$ , the formula is

$$f(x) = \mathcal{FT}^{-1}\{F(k)\}$$

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(k) \exp(ikx) dx$$

Since  $x$  and  $k$  are inversely proportional, the "size" of  $f(x)$  and  $F(k)$  are inversely proportional. What this means is,

- a compact  $f(x)$  will have a broad spectrum.
- a broad  $f(x)$  will have a compact spectrum

## Rectangle function

[Skip to main content](#)



The  $rect_B(x)$  function is a rectangle centered at  $x = 0$  with Height = 1 and Width =  $B$ . The Formula can be written as

$$rect_B(x) = \begin{cases} 0 & \text{if } |x| > B/2 \\ 1/2 & \text{if } |x| = B/2 \\ 1 & \text{if } |x| < B/2 \end{cases}$$

The cell below is a simple function for creating  $rect_B(x)$

## Example Fourier Transform of rect function

Using the FT definition and the  $rect_B(x)$  equation, the FT is

$$\begin{aligned} F(k) &= \int_{-B/2}^{B/2} \exp(-ikx) dx \\ &= -\frac{1}{ik} \exp(-ikx) \Big|_{-B/2}^{B/2} \\ &= -\frac{1}{ik} [\exp(-ikB/2) - \exp(ikB/2)] \end{aligned}$$

Using the complex definition of sine from Euler's formula

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

Our equation for  $F(k)$  can be re-written as

$$\begin{aligned} F(k) &= \frac{2}{k} \frac{\exp(ikB/2) - \exp(-ikB/2)}{2i} \\ &= \frac{2}{k} \sin(kB/2) \end{aligned}$$

[Skip to main content](#)

$$= B \frac{\sin(kB/2)}{kB/2}$$

$$F(k) = B \text{sinc}(kB/2)$$

```
import numpy as np

def rect(x, B):
    """
    create a rectangle function
    returns a numpy array that is 1 if |x| < w and 0 if |x| > w
    w is the rectangle width centered at 0
    x is the number of points in the array
    """

    B = int(B)
    x = int(x)

    high = np.ones(B)
    low1 = np.zeros(int(x/2 - B/2))
    x1 = np.append(low1, high)
    rect = np.append(x1, low1)

    if x > len(rect):
        rect = np.append(rect, 0)
    elif x < len(rect):
        rect = rect[:-1]

    return rect
```

```

%matplotlib inline
%config InlineBackend.figure_format = 'svg'
import matplotlib.pyplot as plt
from IPython import display as disp

import ipywidgets as widgets
from IPython.display import display
slide = widgets.IntSlider()
display(slide)

from IPython.display import display
button = widgets.Button(description="update plot")
display(button)

pi = np.pi
length = 2000
x = np.linspace(-1, 1, length)

def on_button_clicked(b):
    """
    executes function when button is clicked
    """
    B = slide.value * 10
    if B == 0:
        B = 10
    plt.rcParams.update({'font.size': 14})
    plt.rcParams['figure.figsize'] = (12, 1.9)
    plt.yticks([0, 1], ['$0$', '$1$'])
    plt.xticks([-1*B/length, 1*B/length], ['$-B/2$', '$B/2$'])
    plt.plot(x, rect(len(x), B), label=r'$f(x)=rect\_B(x)$')
    plt.axhline(0, color='black', lw=1)
    leg = plt.legend(loc='best', fontsize=14, fancybox=True)
    leg.get_frame().set_linewidth(0.1)
    plt.xlabel('$x$')
    plt.ylim(-0.2, 1.2)
    plt.show()

    plt.yticks([0, 1], ['$0$', '$1$'])
    plt.xticks([-1*pi, 0, 1*pi], ['$-B/2$', '$0$', '$B/2$'])

    k = np.linspace(-1, 1, length)
    plt.plot(x, (B / length) * np.sin(k * B / 2) / (B * k / 2), 'r', label=r'$F(k)=B \sin(kB/2)/(kB/2)$')
    plt.axhline(0, color='black', lw=1)
    leg = plt.legend(loc='best', fontsize=14, fancybox=True)
    leg.get_frame().set_linewidth(0.1)
    plt.xlabel('$k$')
    plt.xlim(-.25, .25)
    plt.show()
    disp.clear_output(wait=True)

button.on_click(on_button_clicked)

```

# testing jupyter-book ok using fnn examples

from [markjay4k/fourier-transform](#) as only use matplotlib inline

## NUMERICAL INTEGRATION

What is numerical Integration

**Numerical** Integration is a way to approximate the value of a definite integral

Unlike Analytic solutions which are exact, closed form solutions

Numerical Integration There are many options for numerically computing definite integrals:

- Trapezoid Rule
- Simpsons Rule
- Gaussian Quadrature
- etc.

Let's look at the Trapezoid Rule

The Trapezoid rule is a way of approximating a definite integral by breaking it up into trapezoids.

Let's say we want to compute the integral

$$\int_0^{\pi} \sin(x) dx$$

We can easily compute it analytically

$$\int_0^{\pi} \sin(x) dx = -\cos(x) \Big|_0^{\pi}$$

$$= -(\cos(\pi) - \cos(0))$$

$$= -(-1 - 1)$$

[Skip to main content](#)

But let's try computing using the trapezoid rule.

Below is a visualization showing  $\sin(x)$  and a trapezoid approximation.

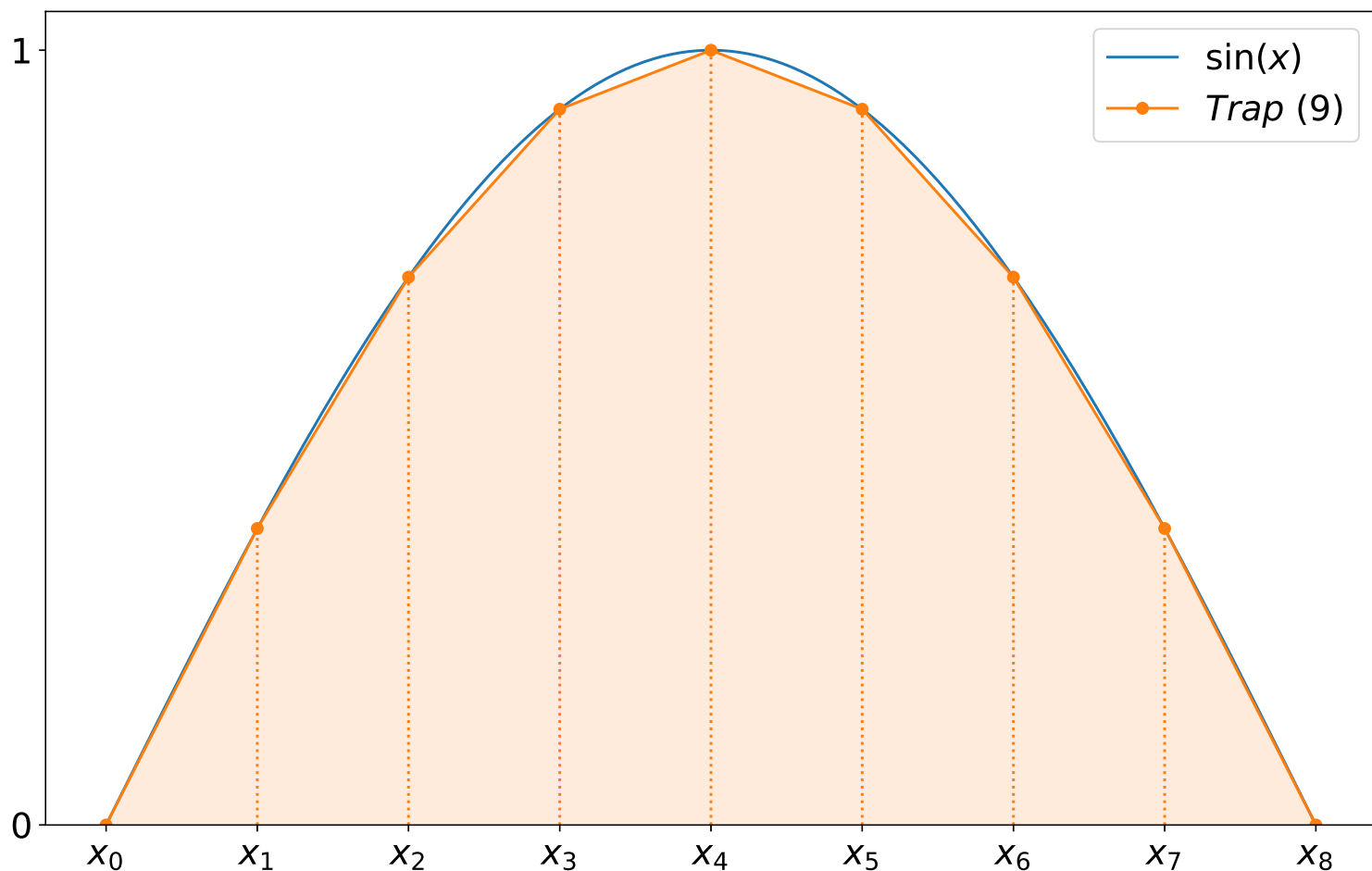
```
import matplotlib.pyplot as plt
import numpy as np
%config InlineBackend.figure_format = 'svg'
plt.rcParams['figure.figsize'] = (13, 8)
plt.rcParams.update({'font.size': 19})

def f(x):
    return np.sin(x)

def trap_plot(n_points):
    x = np.linspace(0, np.pi, 1000)      # continuous
    x_i = np.linspace(0, np.pi, n_points) # discrete

    plt.plot(x, f(x), label=r'$\sin(x)$')
    plt.plot(x_i, f(x_i), '-o', label=r'$Trap$ ({}).format(n_points))
    plt.fill(x_i, f(x_i), color='C1', alpha=0.15)
    plt.vlines(x_i, 0, f(x_i), color='C1', linestyle=':')
    plt.xticks(x_i, [r'$x_{{}}$.format(n) for n in range(n_points)])
    plt.yticks([0, 1], ['$0$', '$1$'])
    plt.legend(loc='best')
    plt.ylim(0, 1.05)
    plt.show()
```

```
trap_plot(9)
```



## Area of a Trapezoid

The area of a trapezoid is the average height times the base



For the example shown, the area is

$$Area = \frac{\overbrace{f(x_{k-1}) + f(x_k)}^{\text{ave height}}}{2} \underbrace{\Delta x}_{\text{base}}$$

trapezoid rule equation

the equation for the trapezoid rule is

$$\int_a^b f(x) dx \approx \frac{f(x_0) + f(x_1)}{2} \Delta x + \frac{f(x_1) + f(x_2)}{2} \Delta x \dots \frac{f(x_{N-1}) + f(x_N)}{2} \Delta x$$

[Skip to main content](#)

Now we can write this as a summation  $\approx \sum_{k=1}^N \frac{f(x_{k-1}) + f(x_k)}{2} \Delta x$

And we can write a function to compute this

$$\approx \sum_{k=1}^N \frac{f(x_{k-1}) + f(x_k)}{2} \Delta x$$

```
def trap(f, x):  
    """  
    computes the integral of f using trapezoid rule  
    """  
    area = 0  
    N = len(x)  
    dx = x[1] - x[0]  
  
    for k in range(1, N):  
        area += (f(x[k - 1]) + f(x[k])) * dx / 2  
  
    return area
```

```
x = np.linspace(0, np.pi, 20)  
trap(f, x)
```

1.9954413183201944

Error vs. number of Trapezoids

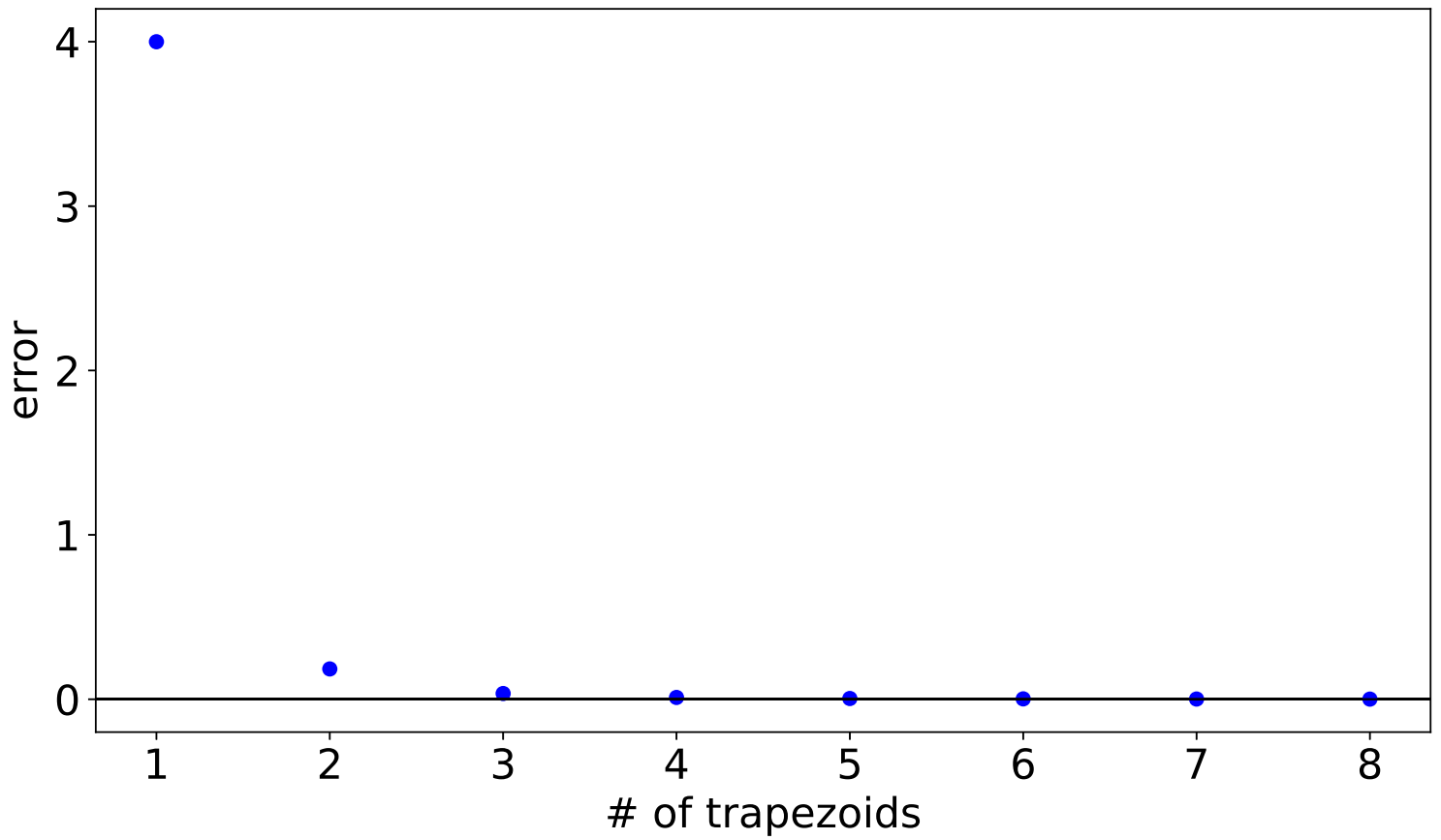
As we increase the number of trapezoids, the approximation gets better ( $error \rightarrow 0$ ).

$error = (actual - approximation)^2$

```
plt.rcParams['figure.figsize'] = (11, 6)  
def plot_error(n_points):  
    for n in range(2, n_points):  
        x = np.linspace(0, np.pi, n)  
        plt.plot(n - 1, (trap(f, x) - 2) ** 2, 'bo')  
  
        plt.axhline(0, color='black', lw=1)  
        plt.xlabel('# of trapezoids')  
        plt.ylabel('error')  
    plt.show()
```

[Skip to main content](#)

```
plot_error(10)
```



testAnimated-Sinc-and-FT-example



```

import numpy as np

def rect(x, B):
    """
    create a rectangle function
    returns a numpy array that is 1 if |x| < w and 0 if |x| > w
    B is the rectangle width centered at 0
    x is the number of points in the array
    """

    B = int(B)
    x = int(x)

    high = np.ones(B)
    low1 = np.zeros(int(x/2 - B/2))
    x1 = np.append(low1, high)
    rect = np.append(x1, low1)

    if x > len(rect):
        rect = np.append(rect, 0)
    elif x < len(rect):
        rect = rect[:-1]

    return rect

```

```

%matplotlib notebook
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# constants and x array
pi = np.pi
length = 2000
x = np.linspace(-1, 1, length)

# create figure and axes
fig, (ax1, ax2) = plt.subplots(2, figsize=(12, 6))

# creating our line objects for the plots
sinc, = ax1.plot(x, np.sin(x), '-b')
box, = ax2.plot(x, np.sin(x), '-r')

def animate(B):
    """
    this function gets called by FuncAnimation
    each time called, it will replot with a different width "B"

    B: rect width

    return:
        sinc: ydata
        box: ydata
    """

    # create our rect object
    f = rect(len(x), B)
    box.set_ydata(f)

    # create our sinc object
    F = (B / length) * np.sin(x * B / 2) / (x * B / 2)
    sinc.set_ydata(F)

    # adjust the sinc plot height in a loop
    ax1.set_ylim(np.min(F), np.max(F))

    # format the ax1 yticks
    plt.setp(ax1, xticks=[-0.25, 0.25], xticklabels=['-1/4', '1/4'],
             yticks=[0, np.max(F)], yticklabels=['0', 'B={:.2f}'.format((B / length))])

    # format the ax2 xticks to move with the box
    plt.setp(ax2, yticks=[0, 1],
             xticks=[-1, -1 * B / length, 1 * B / length, 1], xticklabels=['-1', '-B/2'])

def init():
    """
    initialize the figure
    """

    ax2.set_ylim(-0.2, 1.1)

```

[Skip to main content](#)

```
ax1.axhline(0, color='black', lw=1)
ax2.axhline(0, color='black', lw=1)
plt.rcParams.update({'font.size':14})
```

```
return sinc, box,
```

```
# the FuncAnimation function iterates through our animate function using the steps array
step = 10
steps = np.append(np.arange(10, 1000, step), np.arange(1000, 10, -1 * step))
ani = FuncAnimation(fig, animate, steps, init_func=init, interval=50, blit=True)
plt.show()
```