Numerisk utforskning av spaltegeometrier

Computational essay FYS2130 V25, Jonas Telle

Problemstilling: Hvordan ser diffraksjonsmønsteret fra en dobbeltspalte ut, og kan vi se at det blir svekket ved inkoherens?

Abstract: Vi simulerer lys i interaksjon med ulike totalreflekterende to-dimensjonale geometrier ved å løse bølgelikningen numerisk. Vi vier spesiell oppmerksomhet til dobbelspaltesystemet, og utforsker hvordan både temporal og romlig inkoherens påvirker interferensmønsteret fra denne geometrien visuelt. Simuleringene våre vekker assosiasjoner til andre bølgefenomener, blant annet blåseinstrumenter og vannbølger, og vi gir en kort diskusjon av dette.

Merk: Fordi problemstillingen er av visuell natur og vi undersøker et dynamisk fenomen er animasjoner nødvendige for å gi en fullverdig *besvarelse av problemstillingen. Derfor følger en mappe 'Animations' med denne rapporten. Hvilken animasjon som hører til hvor bør være klart fra filnavnet. Fordi rapporten skal presenteres muntlig, der animasjonene vil vises og kommenteres, er det generelt ikke lagt inn bildeserier i denne pdf-en. Unntaket er bilder som er essensielle for problemstillingen, av diffraksjonsmøsnteret fra dobbeltspalten for koherent og inkoherent lys, som er lagt inn nederst i en egen resultatsdel for enkel sammenlikning.*

Introduksjon

Bølgelikningen beskriver en rekke fysiske fenomener, fra elektromagnetiske felt i far-field til lydbølger i luft og overflatebølger på vannflater. Gode numeriske løsninger av denne kan dermed brukes til å modellere en rekke fysiske fenomener. I moderne fysikk har bølger fått enda større plass, og selv massive enkeltpartikkler kan modelleres som en bølge når de ikke interagerer med omgivelsene sine. Et av de mest kjente eksperimentene i fysikkens historie, nemlig dobbeltspalte-eksperimentet, illustrerer at bølger interagerer med ulike geometrier på måter som, for de fleste av oss, er mindre intuitive enn klassiske partikler, som vi er godt kjent med fra hverdagen. Jeg vil dykke inn i bølgene ved å studere dette mystiske mønsteret som har tatt nattesøvnen fra utallige fysikere. Når jeg endelig har klart å simulere det, vil jeg forsøke å viske det ut igjen ved hjelp av et særegent bølgefenomen, nemlig inkoherens. Vi legger hele veien spesiell vekt på fleksibel implementering, slik at arbeidet lett kan utvides til de utallige andre fenomener som bølgelikningen beskriver, og drøfter til slutt flere mulige utvidelser og videre anvendelser.

Teori

Diskretisering av bølgelikningen

I far-field har Maxwells likninger planbølge-løsninger, så vi kan med god tilnærming modellere lys med bølgelikningen

$$\frac{1}{c^2}\frac{\partial^2 E}{\partial t^2} = \nabla^2 E.$$

Vi diskretiserer derfor denne i 2 dimensjoner. Vi må visualisere i 2D fordi lysbølgene er transverse, og det er tydeligst å vise magnituden med et fargeplott, men vi kunne alternativt ha løst likningen i 3D og vist et utsnitt om vi ville være mer nøyaktige eller påvise Arago's flekk. Grunnen til at dette ikke er gjort er fordi det er svært regnetungt å legge til en dimensjon, men programmet blir ikke noe mer komplisert - diskretiseringen i 3D følger den samme prosedyren vi nå skal gjennomgå:

Fra definisjonen av den deriverte kan vi tilnærme

$$\frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

som gir

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{\frac{f(x+\Delta x)-f(x)}{\Delta x} - \frac{f(x)-f(x-\Delta x)}{\Delta x}}{\Delta x} = \frac{f(x+\Delta x) + f(x-\Delta x) - 2f(x)}{\Delta x}, \quad (1)$$

der Δx er en liten steglengde for en diskretisering av en kontinuerlig variabel x og f er en funksjon.

Vi innfører notasjonen $E^n_{i,j}$ for $E(t_n, x_i, y_j)$ der indeksene angir hvor i diskretiseringen av variabelens kontinuum vi er. Det betyr ganske enkelt at x_i er $x[i]$ hvis x er en np.linspace over x-verdier. Vi innfører også

$$\rho = \left(c\frac{\Delta t}{\Delta x}\right)^2 < C = 0.5$$

der ulikheten med konstant C er Courant-Friedrichs-Lewy-betingelsen (som vi i arbeidet med denne oppgaven flere ganger kom til skade for å overstige, hvilket ga overflow-errors), som sier noe om numerisk stabilitet i systemet. $C = 0.5$ er valgt på oppfordring fra [1], der man også finner en gjennomgang av bølgelikning-diskretiseringen (om i en litt annen form enn her, fordi de har med et dempningsledd). Med denne notasjonen, og utregningen fra likning (1) får vi

$$\frac{1}{c^2}\left[\frac{E^{n+1}_{i,j} - 2E^n_{i,j} + E^{n-1}_{i,j}}{\Delta t^2}\right] = \frac{E^n_{i+1,j} - 2E^n_{i,j} + E^n_{i-1,j}}{\Delta x^2} + \frac{E^n_{i,j+1} - 2E^n_{i,j} + E^n_{i,j-1}}{\Delta y^2}.$$

Løser vi denne for tidssteg $n+1$ gitt tidssteg n og $n-1$, der vi krever $\Delta x = \Delta y$ for enkelhets skyld, får vi endelig

$$E^{n+1}_{i,j} = \rho\left(E^n_{i+1,j} + E^n_{i-1,j} + E^n_{i,j-1} + E^n_{i,j+1} - 4E^n_{i,j}\right) + 2E^n_{i,j} - E^{n-1}_{i,j}.$$

Grensebetingelser

For å løse bølgelikningen trenger vi også initial- og rand-betingelser. Initialbetingelsene er i vårt tilfellet naturlig valgt til null overalt. Hva randbetingelsene angår har vi to valg: Dirichlet-betingelser, som innebærer å feste verdien i bestemte punkter, eller von Neumann-betingelser, som matematisk innebærer å sette den deriverte lik null i et punkt. Vi klarer oss strengt tatt med Dirichlet-betingelser her, men von Neumann-betingelser kan være nyttige om man vil unngå refleksjon/interferens fra plotteområdet og er en naturlig utvidelse av arbeidet vårt. Numerisk er det ikke vanseklig å legge til disse, men det krever noen del if-tester i løsningsløkka som gjør koden litt styggere, så vi utvider heller området vi gjør beregningene på og plotter bare en liten del. Dette viser seg å gi gode resultater. En annen anvendelse av von Neumann-betingelser ville være om man vil modellere delvis refleksjon/flater med ulik impendans, men dette er mer relevant for instrumentsimuleringer enn for spaltegeometriene vi skal utforske. Vi lager altså ganske enkelt en stor boks med totalreflekterende vegger (null E-felt, alltid).

Men for øyeblikket vil det ikke skje noe som helst i denne boksen: For å få noen spennende resultater må vi også generere bølgene! Dette gjør vi ved å variere initialbetingelsene langs den ene siden av boksen vår harmonisk i tid. Disse svingningene vil forplante seg gjennom boksen og simulere lys som stømmer fra en kilde mot en skjærm. Med denne metoden kan vi også simulere mer realistiske situasjoner med inkoeherens, både i tid og rom - mer om dette til slutt.

Spaltegeometrier

Vi kan nå konstruere hindringer i bølgenes skuddlinje. Vi starter med en enkeltspalte, som gir en illustrasjon av Huygens prinsipp ved at lyset synes å forplante seg fra hindringen som om det var en ny kilde i spalten. Vi kan også utforske en pinne-hindring, altså inversen til enkeltspalten, en gitterstruktur og en sirkulær hindring. Programmet vi skriver er fleksibelt, og man kan enkelt legge til hvilke hindringer man enn måtte ønske. Vi poengterer også at programmet vårt kan brukes til å simulere luftbølger i instrumenter, men at man i såfall gjerne kunne implementert von Neumann-betingelser for å få ulike impendanser i instrumentets ender.

Dobbeltspalten og koherens

Vi går så over til oppgavens fokus; dobbeltspalten. Her forventer vi, gitt at bølgelengden er på størrelsesorden med spaltevidden, et interferensmønster. Vi kan deretter forsøke å forstyrre dette mønsteret ved å redusere bølgens koherens; vi gjør dette temporalt ved å superimponere ulike faseforsøvede harmoniske bølger, og romlig ved å dele opp kilden i flere små kilder med litt forsøvet fase.

Numerikk

Vi implementerer ideene diskutert over. Først trenger vi noen vanlige Python-biblioteker:

In [1]:
```
# Basic imports
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.animation as animation
import seaborn as sns
from numba import jit

# Settings
np.random.seed(42)
sns.set_style(style='white')

# Interactive plots in ipynb (eg. animation)
%matplotlib ipympl
```

Løsning av den diskretiserte bølgelikningen i 2D

Vi løser så bølgelikningen med en klasseimplementering. Merk at tidssteg-funksjonen bruker Just-in-time-compilation, som gjør løkkene mellom 10 og 1000 ganger raskere. Vi implementerer grensebetingelsene på standard vis (fra FYS1120) med en matrise B med verdi np.nan der beregningene skal foregå og flytverdier der vi har betingelser.

In [2]:
```
# Class helper, jit does not like python-classes/self argument
@jit
def timestep(Bn, rho, u, u_old=None):   # Advances solution by one time step
    I, J = u.shape
    u_new = np.empty((I,J))

    if u_old is None:                   # At first step, no U[n-1]
        u_old = u

    for i in range(I):
        for j in range(J):

            if np.isnan(Bn[i,j]):
                spatial_term = u[i+1,j] + u[i-1,j] + u[i,j+1] + u[i, j-1] -
4*u[i,j]
                u_new[i,j] = rho*spatial_term + 2*u[i,j] - u_old[i,j]

            else:
                u_new[i,j] = Bn[i,j]

    return u_new


# Solver
class WaveEq:
    def __init__(self, B, dt, dx, c=1, u0=None):
```

```
        """
        B is the time dependent boundary conditions (N by I by J array)
            and should be NaN except at boundary condintions.
        dt is the time-step-length.
        dx is the space-step-length.
        c is the speed of light.
        u0 is the initial conditions (I by J array);
            if None the field is initially zero everywhere.
        """
        self.rho = (c*dt/dx)**2
        assert self.rho < .5, "Ustable system, reduce time step or increase space
step."

        self.B = B
        req = not np.any(np.isnan(B[:,[0,-1],:])) or np.any(np.isnan(B[:,:,
[0,-1]]))
        assert req, "Boundary conditions required."

        self.dt, self.dx = dt, dx
        self.N, self.I, self.J = B.shape

        if u0 is None:
            u0 = np.zeros((self.I, self.J))
        self.u0 = u0


    def solve(self):
        B, rho = self.B, self.rho

        U = np.empty_like(self.B)           # solution array
        U[0] = self.u0                      # initial conditions
        U[1] = timestep(B[0], rho, U[0])    # first step separately, no U[n-1]

        for n in range(1, self.N-2):        # solving
            U[n+1] = timestep(B[n], rho, U[n], U[n-1])

        self.U = U                          # solution as class variable


    def animate(self, interval=10, speedup_factor=5, filename=None): # Animates
the solution

        J = self.J
        J4 = J // 4
        U = self.U[:,5:,J4:J-J4+1]          # only plot half of the box due to
edge effects

        # Matplotlib animation
        fig, ax = plt.subplots()
        cax = ax.imshow(U[0].T, cmap="viridis", origin="lower") # heatmap,
conventions: transpose for (t, x, y) and
        fig.colorbar(cax)                                       # lower origin for
upwards y-axis

        def update(frame): # update rule for FuncAnimation
            cax.set_data(U[frame].T)
```

```
            return [cax]

        # The speedup_factor is the number of frames we skip - it makes the export
way faster with fewer frames, and
        # since the framerate is fixed, also the gif runs faster. Blit=True only
redraws the changed pixels for efficiency.
        anim = animation.FuncAnimation(fig, update, frames=range(0, len(U),
speedup_factor), interval=interval, blit=True)

        if filename is not None:
            print("The animation is finished, but it may take a while to export!")
            anim.save(filename, writer=animation.PillowWriter(fps=30))
        else:
            print("Animations are buggy in IPYNB. I recommend using a .py-file or
making an export.")
            plt.show()
```

Grensebetingelser

Vi skriver deretter en funksjon som lager boksen med grensebetingelser. Vi husker at vi senere skal prøve andre lyskilder, så vi forsørger at implementeringen er fleksibel:

In [3]:
```
def harmonic_source(A, f, t, J):            # coherent source
    return A * np.sin(2 * np.pi * f * t)


def make_B(N, I, J, geometry=5, source=harmonic_source, dt=.001, a=5, d=20, A=1,
f=100, R=7):
    """
    Creates the N by I by J condition matrix B.

    geometry:   1 = Single barrier, 2 = Single slit, 3 = Tight lattice,
                4 = Circle,         5 = double slit (default)
    source:     func(A, f, t, J) -> array of length J
                    Generates the wave at time n*dt
    Optionals:  dt is time step, a is slit width, d is distance between slits,
                    A is source amplitude, f is source frequency, R is radius
    """
    B = np.full((N, I, J), np.nan)          # initialising

    # Time dependent (source)
    for n in range(N):
        t = n*dt                            # time at step n
        B[n, 0,  :] = source(A, f, t, J)    # left boundary oscillating

    # Time independent (reflecting walls - can use von neumann for non-reflective,
    B[:, -1, :] = 0   # but this reqires a slight modification of the solver)
    B[:, :,  0] = 0
    B[:, :, -1] = 0

    J2, J4, I3 = J//2, J//4, I//3           # useful short-hands for creating
geometries

    if geometry == 1:                       # Single barrier, width 2*d
```

```
        B[:, I3, J2-2*d:J2+2*d+1] = 0

    if geometry == 2:                       # Single slit, width 2*d
        B[:, I3, :] = 0
        B[:, I3, J2-2*d:J2+2*d+1] = np.nan

    if geometry == 3:                       # Tight lattice
        B[:,I3,:J4+a+1] = 0
        B[:,I3,J-J4-a:] = 0
        B[:,I3,J4+a:J-J4-a+1:a] = 0

    if geometry == 4:                       # Circle
        for i in range(I):
            for j in range(J):
                if (i - I//3.5)**2 + (j - J//2)**2 < R**2:
                    B[:, i, j] = 0

    if geometry == 5:                       # Double slit
        B[:, I3, :] = 0
        B[:, I3, J2-a-d:J2+a+1-d] = np.nan
        B[:, I3, J2-a+d:J2+a+1+d] = np.nan

    return B
```

Vi kan nå velge noen parametere...

In [4]:
```
# Parameters
N = 1_200                   # number of timepoints
I, J = 220, 200             # number of X and Y points
dt, dx = .001, .005         # timestep and spacestep
T, X, Y = dt*N, dx*I, dx*J  # Total time, X- and Y-distance
```

...og starte plottingen! Vi kommer til å lage en rekke animasjoner, så vi lager en wrapper for denne prosessen:

In [5]:
```
def make_anim(filename, option=5, source=harmonic_source, speedup_factor=5,
interval=20, N=N, I=I, J=J, dt=dt, a=5, d=10, A=10, f=100, R=25):
    B = make_B(N, I, J, option, source, dt, a, d, A, f, R)

    wave = WaveEq(B, dt, dx)
    wave.solve()
    wave.animate(interval, speedup_factor, filename)
    plt.close() # Remove if you want to display the animation in the notebook
(this is somewhat buggy, though)
```

Vi utfører et eksempel av hver geometri, og avslutter med dobbeltspalten (figur 1). Noen flere variasjoner på dobbelspalten med ulike parametere finnes også i mappa 'Animations', og leseren er dessuten oppfordret til selv å utforske parametere ved å leke med de frivillige parameterene i funskjonskallet til make_anim under.

In [6]:
```
filenames = ['single_barrier', 'single_slit', 'tight_lattice', 'circle',
'double_slit']
```

```
for i in range(5):
    B = make_anim(filenames[i] + '.gif', i+1, speedup_factor=10)
```

```
The animation is finished, but it may take a while to export!
The animation is finished, but it may take a while to export!
The animation is finished, but it may take a while to export!
The animation is finished, but it may take a while to export!
The animation is finished, but it may take a while to export!
```

(**En liten parentes om fløyter:** Gitter- og spalte-simuleringene vekker assosiasjoner til bevegelsen av lydbølger i et blåseinstrument, for eksempel en fløyte, der bølgene går frem og tilbake i en lukket sylinder og transmitteres hver gang de når den åpne enden. Hvis vi ville lage en bedre simulering av et slik instrument kunne vi bruke en von Neumann-betingelse med en koeffisient på overgangs-overflaten for å skape en lav-impendans-overflate. Det kunne da også være gunstig å bruke von Neumann-grensebetingelser langs kantene av simuleringen for å unngå refleksjoner her, men dette er ikke hensikten med dette prosjektet. Det er likevel interessant å se hvor generell bølgelikningen er, og at tilsynelatende ulike fysiske systemer likevel kan beskrives ved hjelp av den.)

Inkoherens

Vi returnerer til dobbeltspaltesituasjonen for å utforske hvorvidt inkoeherent lys ødelegger dobbeltspaltemønsteret. Vi simulerer inkoherens ved å superimponere sinusbølger med tilfeldige fasebidrag i randbetingelsene som genererer bølgen:

In [7]:
```
num_sources = 50
def temporal_incoherent_source(A, f, t, J, num_sources=num_sources):

    phases = np.random.uniform(0, 2*np.pi, num_sources)         # Random phase
    return np.sum([A*np.sin(2*np.pi*f*t + phi) for phi in phases])

# Animate - mostly same parameters, amplitude scaling for visual aid
B = make_anim("double_slit_temporal.gif", source=temporal_incoherent_source, A=.1,
f=100)
```

```
The animation is finished, but it may take a while to export!
```

Med lav temporal koeherens ser vi at interferensmønsteret blir betydelig mer utydelig (figur 2).

Vi tester så romlig inkoherens ved å dele opp kilden i flere kilder med ulike faser. Først bruker vi få, lange kilder med stor variasjon mellom hver kilde:

In [8]:
```
# Few sources, large variation
num_sources = 7 # This number has a massive influence on the wave with this model
spacing = J // num_sources

@jit # Alternatively, you could vectorize the loop, but this is as fast and much
faster to code
def spatial_incoherent_source(A, f, t, J, spacing=spacing):
    source = np.empty(J)
```

```
    for i in range(J):

        if i % spacing == 0:
            phi = np.pi*np.random.uniform(0, 2*np.pi)

        source[i] = A*np.sin(2*np.pi*f*t + phi)

    return source

# Animate
B = make_anim("double_slit_spatial_uniform.gif", source=spatial_incoherent_source,
A=1, f=50)
```

```
The animation is finished, but it may take a while to export!
```

Vi ser at interferensmønsteret nærmest forsvinner (figur 3). Jeg blir også slått av hvor mye disse bølgene kan likne på bølger på en vannflate, som kan tyde på at enkelte typer overflatebølger på vann kan modelleres på denne måten!

For meg er likevel ikke denne situasjonen den mest interessante, fordi lyset så åpenbart er inkoherent også før barrieren. Jeg ønsker heller en situasjon der bølgefrontene synes å ligge nokså parallellt, men der interferensmønsteret likevel er tydelig forstyrret. Lyskilder er jo mer naturlig produsert av mange normalfordelte fenomener som ligger tett i forhold til bølgelengden (i solide stoffer ligger atomene med avstand på størrelsesorden 0.1nm, mens synlig lys som kjent har bølgelengde på flere 100nm). Dette motiverer et forsøk på en mer realistisk simulering, der vi bruker veldig mange flere kilder med normalfordelte og svært like fasebidrag:

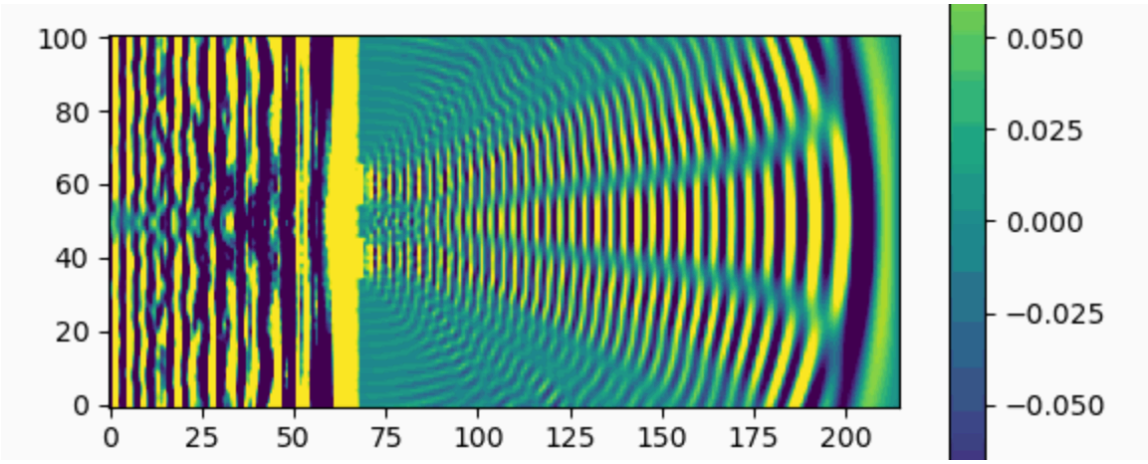
In [9]:
```
# Many sources, small variation
num_sources = 50
spacing = J // num_sources
k = .3                          # weight coefficient

@jit
def spatial_normal_incoherent_source(A, f, t, J, spacing=spacing, k=k):
    source = np.empty(J)
    for i in range(J):
        if i % spacing == 0:
            phi = k*np.pi*np.random.normal()
        source[i] = A*np.sin(2*np.pi*f*t + phi)

    return source


# Animate
B = make_anim("double_slit_spatial_normal.gif",
source=spatial_normal_incoherent_source, A=1, f=50)
```

```
The animation is finished, but it may take a while to export!
```

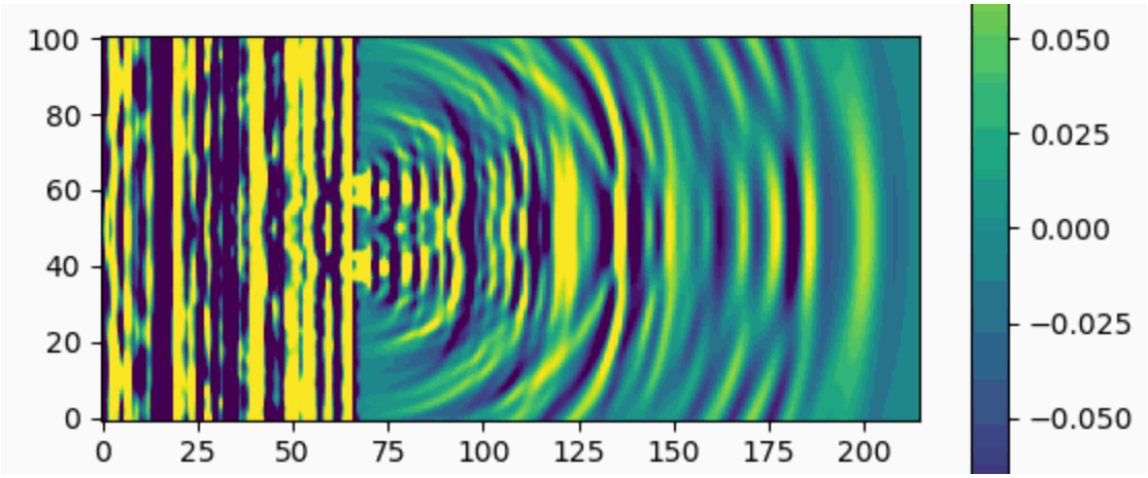
Her ser bølgefrontene noe mer rette ut, men likevel er interferensmøsteret svekket (figur 4). Kult!

Resultater

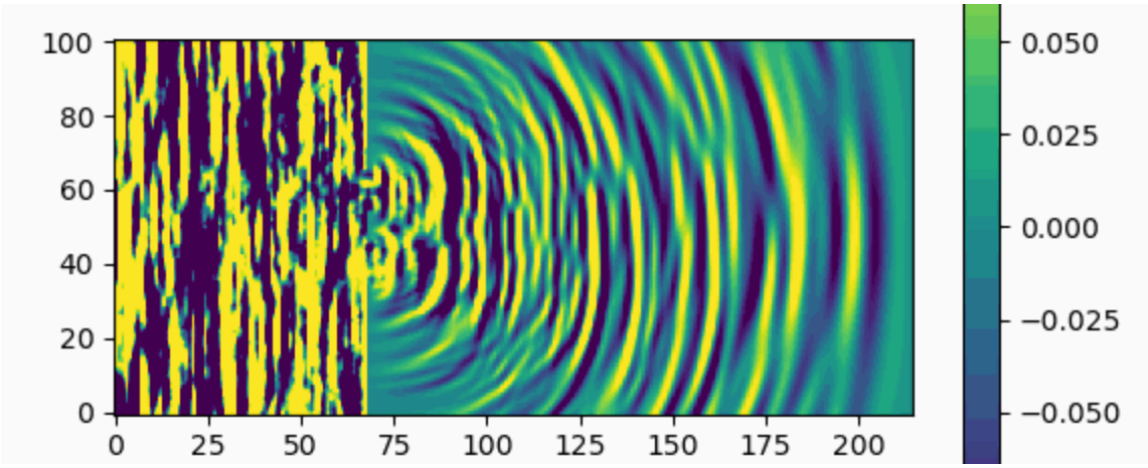
Denne seksjonen er tiltenkt lesere som kun har tilgang til pdf-versjonen av dette dokumentet og ikke kan se animasjonene underveis, og inneholder bare bilder som er essensielle for besvarelsen av problemstillingen. En samling av animasjoner finnes i mappa 'Animations' som skal følge rapporten.



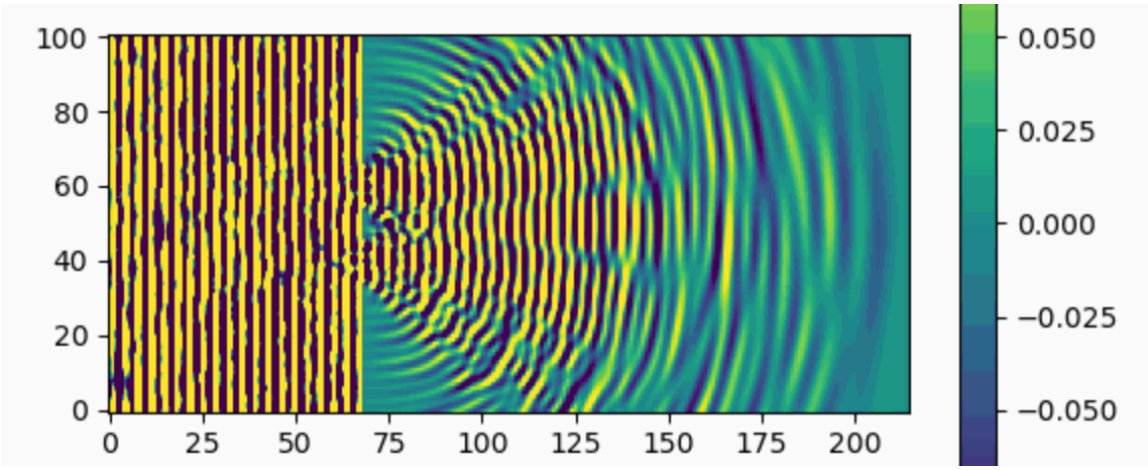
Figur 1: *Dobbelspaltediffraksjonsmønster for koherente bølger. Vi ser helt tydelig hvordan bølgene interferer og skaper stasjonære stråler i feltet.*



Figur 2: *Dobbelspaltediffraksjon med temporal inkoherens. Vi ser fremdeles svake spor etter diffraksjonsmøsteret i figur 1, men det er åpenbart svekket!*



Figur 3: *Dobbelspaltediffraksjon med romlig inkoherens fra få kilder med uniformfordelt fase (stor variasjon mellom kildene). Igjen er diffraksjonsmønsteret godt gjemt.*



Figur 4: *Dobbelspaltediffraksjon med romlig inkoherens fra mange kilder med dempet normalfordeltfordelt fase (liten variasjon mellom kildene). Merk først at området før barrieren ser nokså koherent ut (sammenlikn gjerne med figur 1, men dette er ikke klart uten animasjonen), men at diffraksjonsmønsteret har en svært ulik form fra figur 1, der mønsteret oppstår umiddelbart. I stedet oppstår det et halvtydelig diffraksjonsmønster (tilsynelatende flere superimponerte diffraksjonsmønstere, om man sammenlikner med figur 1) etter noe tid og nærme spalten, men det når ikke fram til skjermen i simuleringstiden.*

Konklusjon

Vi har simuleret lys i interaksjon med ulike totalreflekterende to-dimensjonale geometrier ved å løse bølgelikningen numerisk. Spesielt har vi sett tydelig hvordan diffraksjonsmønsteret fra dobbelspaltegeometrien ser ut for noen ulike parametere, og at dette svekkes betydelig ved både temporal og romlig inkoherens. Dette kan også være tilfellet når det ikke umiddelbart er åpenbart fra bølgefrontene før dobbeltspalten at lyset er inkoherent. Problemstillingen er altså tydelig bekreftet!

Likevel er det flere mulige forbedringer og utvidelser som kunne styrket prosjektet. Å utvide til tre dimensjoner krever bare en liten modifikasjon i programmet, men en god del regnekraft (eller eventuelt svært mye tålmodighet). Før man gjør denne utvidelsen bør man bruke von Neumann-grensebetingelser langs randen av boksen for å unngå å måtte simulere et større område enn man plotter, da den ekstra dimensjonen vil øke kostnaden av bortkastet simulering betydelig. Dette er uansett en interessant forbedring om man vil åpne for simuleringer av andre bølgefenomener, slik som instrumenter. I 3D blir barrierene kanskje mer interessante, og man kan se etter Aragos flekk! Det hadde også vært spennende å forsøke å måle intensiteten av bølgene mot skjærmen (bakerste vegg), og man kunne da sammenliknet disse resultatene med luminositetskurver som kan beregnes numerisk (men basert på analytiske utrykk fra Huygens prinsipp, altså uten modellering - et eksempelprogram finnes i nettressursene tilknyttet [2]). Dette er nok litt vanskeligere enn det høres ut, for den numeriske modellen vår har en tendens til å tape energi på mystisk vis, men om man kom seg over kneika og fikk gode resultater ville man ha en kvantitativ validering av modellen utover det visuelle. Til slutt kunne man selvsagt utvidet til flere geometrier, og dessuten sendt bølgene gjennom diverse hiderløyper eller labyrinter.

Deklarasjon av bruk av generativ kunstig intelligens

I dette vitenskapelige arbeidet har generativ kunstig intelligens (KI) ikke blitt benyttet.

Referanser

[1]: Adams, H. V. (n.d.). *Discretizing the 2D Wave Equation*. https://vanhunteradams.com/DE1/Drum/Discretization.html. Retrieved 04.04.2025. Authors association: Cornell University.

[2]: Vistnes, A. I. (2018). *Phyiscis of Oscillations and Waves: With use of Matlab and Python*. Springer Cham. https://doi.org/10.1007/978-3-319-72314-3.