Simulação de ondas 2D com Python

Descrição do Fenómeno Escolhido

Com este projeto computacional pretende-se mostrar como, usando Python, é possível simular e visualizar ondas a 2 dimensões usando o Método das Diferenças Finitas. Também veremos alguns exemplos de interferência, mais uma vez, com enfoque na visualização do fenómeno.

Descrição das Ferramentas Utilizadas

Para este projeto usou-se a linguagem Python, em Jupyter Notebooks. Os Jupyter Notebooks permitem integrar texto com código de uma forma interativa para a partilha fácil e intuitiva entre diferentes pessoas. Este relatório é só a parte textual do notebook que foi criado para este projeto. Escolheu-se Python por ser uma linguagem pragmática, com excelentes bibliotecas e muito potencial de aprendizagem.

Descrição do Método Utilizado - Método das Diferenças Finitas

Para este método partimos da equação das ondas. Como vimos durante o semestre, a equação das ondas é uma consequência da relação de dispersão, mas a sua forma torna-a muito útil para esta simulação. Para este projeto, irei considerar ondas acústicas, ou seja, a onda é dada pela pressão p. Em 2D a equação acústica de ondas é dada por:

$$\frac{\partial^2 p(x,y,t)}{\partial t^2} \;=\; v^2 \left(\frac{\partial^2 p(x,y,t)}{\partial x^2} + \frac{\partial^2 p(x,y,t)}{\partial y^2} \right)$$

No fundo, o Método das Diferenças Finitas é uma aproximação no cálculo das derivadas. Olhemos para três definições diferentes de derivada:

$$f'(x) = \lim_{dx \to 0} \frac{f(x+dx) - f(x)}{dx}$$

$$f'(x) = \lim_{dx \to 0} \frac{f(x) - f(x-dx)}{dx}$$

$$f'(x) = \lim_{dx \to 0} \frac{f(x+dx) - f(x-dx)}{2dx}$$

Para obter uma aproximação destes cálculos, esquecemos o limite e calculamos o valor da derivada com um dx "pequeno". O que "pequeno" quer dizer concretamente veremos mais tarde. Por enquanto, observemos que este método de cálculo da derivada é uma aproximação e, como tal, podemos ter uma noção da ordem de grandeza do erro. Para isso, expandimos o cálculo com Séries de Taylor:

$$f(x+dx) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!} dx^n$$

Para a primeira definição de derivada, podemos rearranjar os termos e obter:

$$\frac{f(x+dx) - f(x)}{dx} = f'(x) + O(dx)$$

Ou seja, o erro é em ordem a dx. Repetindo a matemática para as outras duas definições, o erro na segunda é igual, mas, surpreendentemente, na terceira definição o erro vai em ordem a dx^2:

$$\frac{f(x+dx)-f(x-dx)}{2dx} = f'(x) + O(dx^2)$$

Logo, à partida, esta será a melhor aproximação a utilizar para a primeira derivada.

No entanto, precisamos de aproximações para derivadas de segunda ordem. Seguindo uma lógica semelhante, podemos chegar à conclusão de uma boa aproximação para a segunda derivada, usando os mesmos três pontos: f(x+dx), f(x) e f(x-dx):

$$f''(x) \approx \frac{f(x+dx)-2f(x)+f(x-dx)}{dx^2}$$

O que tudo isto significa é que estamos a discretizar o espaço e o tempo. Em vez de ser contínuo, temos uma grelha de quadrados de lado dx e dy, que varia no tempo em pequenos passos dt, e a cada quadrado está associado um valor de pressão:

$$p(x,y,t) \ \rightarrow \ p^n_{i,j} \ = \ p(idx, jdy, ndt) \ .$$

Ou seja, estamos a fazer o oposto daquilo que fizemos quando passámos de osciladores discretos para oscilações contínuas.

Para finalizar esta já longa discussão, observemos que o que pretendemos é saber a evolução da pressão com o tempo em cada quadrado. Podemos sabê-lo aplicando as aproximações à equação de ondas:

$$\frac{p^{n+1}_{i,j} - 2p^n_{i,j} + p^{n-1}_{i,j}}{dt^2} \ = \ c^2 \left(\frac{\partial^2 p(x,y,t)}{\partial x^2} + \frac{\partial^2 p(x,y,t)}{\partial y^2}\right)$$

em que:

$$\begin{aligned}
\frac{\partial^2 p(x,y,t)}{\partial x^2} &= \frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{dx^2} \\
\frac{\partial^2 p(x,y,t)}{\partial y^2} &= \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{dy^2} \ .
\end{aligned}$$

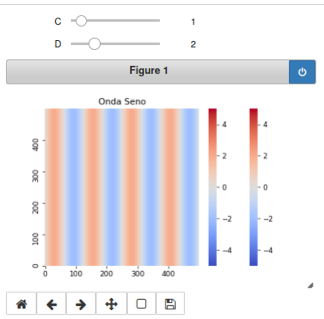
Note-se que para efeitos de simplicidade vamos utilizar dy=dx

Destas equações conseguimos retirar a pressão no "futuro" (t = (n+1)dt),a partir da pressão "agora" em n e da pressão no "passado", em n-1. Iterando com um ciclo for, podemos assim ver a evolução da pressão com o tempo.

Apresentação e Discussão de Resultados

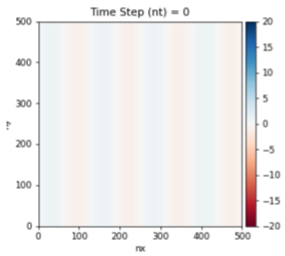
Comecemos com um primeiro exemplo simples de simulação de ondas com Python, de uma simples função seno cuja amplitude e frequência podemos alterar e visualizar.

$$f(x,y) = D * sin(0.05 * C * x)$$

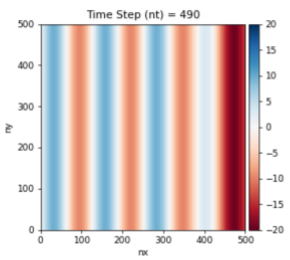


Note-se que o método que se está a usar neste relatório para a visualização destas funções 3D são estes *heat maps*, em que as duas dimensões representam o espaço discretizado xy e a cor representa a pressão no ponto (x,y) em questão.

O próximo exemplo de simulação é mais uma vez pegando na mesma onda seno simples para descrever as condições iniciais do sistema, e utilizar o Método das Diferenças Finitas para simular a sua evolução com o tempo. Estado inicial:



Após 500 *time steps*:



Podemos observar já nesta simulação aquele que é um defeito deste tipo de simulações: erros numéricos que podem fazer a simulação divergir do pretendido. Nesta simulação da propagação a partir de uma função seno vemos uma mancha vermelha na direita a aparecer, que parece claramente ser um erro da simulação. Neste caso, talvez tenha a ver com a falta de informação para o cálculo das derivadas nas fronteiras e da aproximação se tornar mais grosseira aí. De qualquer forma, a análise matemática deste tipo de erros vai para além do propósito deste projeto, mas deixo aqui dois critérios simples que procurei cumprir.

Número de pontos por comprimento de onda

Este critério é bastante intuitivo, tendo em conta a sua semelhança com outros critérios que estudámos durante o semestre para fazer aproximações semelhantes, e indica que quanto mais pontos dx ou dy houver por comprimento de onda λ, mais estável será a simulação. Este é um dos critérios que permite determinar o quão "pequeno" devem ser dx e dy, ou seja, são pequenos relativamente ao comprimento de onda da onda que estivermos a estudar.

Critério de Courant

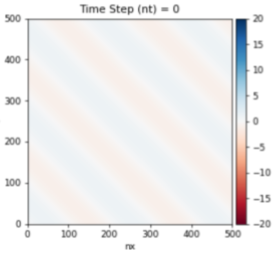
O critério de Courant, derivado através da análise matemática deste método, define-se como:

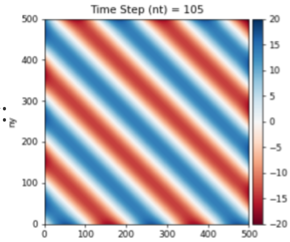
$$\epsilon = c\frac{dt}{dx} \leq 1$$

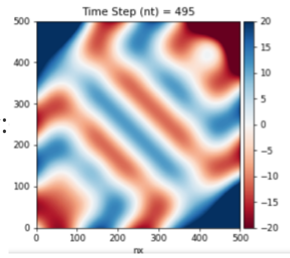
Com esta informação podemos calcular o tamanho de dt para uma simulação estável, que está portanto relacionado com dx e com a velociade de propagação de ondas no meio em estudo.

O próximo exemplo será semelhante, mas com uma função um pouco mais interessante:

$$f(x, y) = sin(0.03 * (x + y))$$

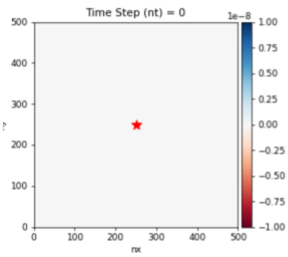
Aspeto inicial: 

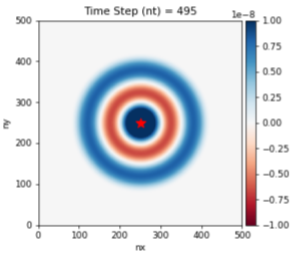
Após cerca de 100 *time steps*: 

Após 500 *time steps*: 

Passemos a uma análise diferente, onde também iremos abordar o fenómeno de interferência. Desta vez, inicializamos a pressão a 0 em todo o nosso espaço, e adicionamos uma fonte de sinal no centro. Assim, podemos simular a forma como o sinal se propaga. Para começar olhemos para a função simples:

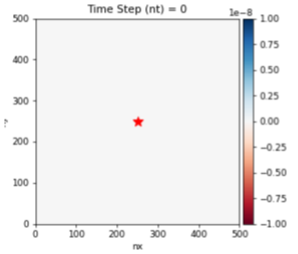
$$f(t) = sin(20 * t) * 10^{-2}$$

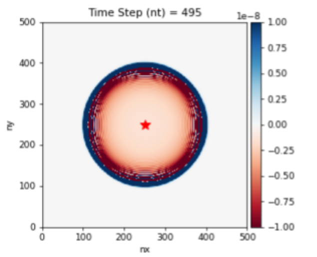
Aspeto inicial: 

Aspeto final: 

De seguida, um exemplo semelhante mas com uma função esquisita o suficiente para valer a pena ser simulada:

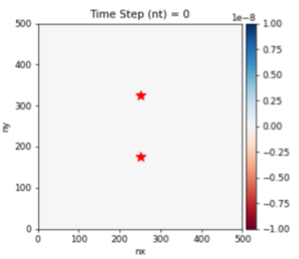
$$f(t) = -8 * (t - t0) * f0 * e^{-(4*f0)^2*(t-t0)^2}$$

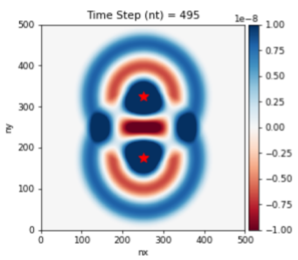
Aspeto inicial: 

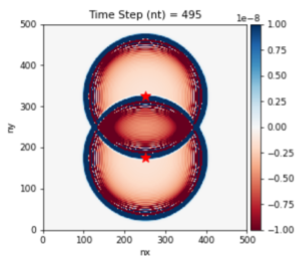
Aspeto final: 

Interferência

Por fim, podemos usar esta simulação simples para observar outro fenómeno importante - a interferência entre duas ondas. Visto que a interferência não é mais do que a soma de dois sinais/ondas, a simulação é computacionalmente muito simples. Usando as duas funções anteriores como os nossos sinais, colocamos desta vez duas fontes diferentes no espaço.

Aspeto Inicial: 

Aspeto Final Dois Senos: 

Aspeto Final Duas Gaussianas: 

Conclusão

Com este projeto, é possível demonstrar uma introdução à simplicidade com que o Python nos permite fazer simulações computacionais interessantes e poderosas, e olhar de uma outra forma ao conteúdo aprendido nesta cadeira ao longo do semestre.