CDS: Numerical Methods - Assignment week 6

Solutions to the exercise have to be handed in via Brightspace in form of one or several executable Python scripts (*.py) which run without any errors. The deadline for the submission is **Thursday Mar. 19, 13:30**. Feel free to use the Science Gitlab repository to submit your solutions.

1 Dynamic Behavior from Parabolic and Hyperbolic PDEs

In the following you will derive and implement finite-difference methods to study generalized hyperbolic partial differential equations and the (parabolic) Schrödinger equation in 1D.

1.1 Hyperbolic PDEs

(a) Finite-difference hyperbolic PDE solver: Our aim is to implement a Python function to find the solution of the following PDE:

$$\frac{\partial^2}{\partial t^2}u(x,t) - \alpha^2\frac{\partial^2}{\partial x^2}u(x,t) = 0, \qquad 0 \le x \le l, \qquad 0 \le t$$

with the boundary conditions

$$\begin{aligned} u(0,t) = u(l,t) &= 0, && \text{for } t > 0 \\ u(x,0) &= f(x) && \\ \frac{\partial}{\partial t}u(x,0) &= g(x) && \text{for } 0 \le x \le l. \end{aligned}$$

By approximating the partial derivatives with finite differences we can recast the problem into the following form:

$$\vec{w}_{j+1} = \mathbf{A}\vec{w}_j - \vec{w}_{j-1},$$

where $\vec{w}_j$ is a vector in the discretized spatial coordinate $x_i = ih$ with $i = 0, 1, \ldots, m$ and $h = l/m$ and j labels the descretized time steps $t_j = jk$ with $j = 0, 1, \ldots$. Here, $\mathbf{A}$ is a $(m-1)\times(m-1)$ tri-diagonal matrix defined by:

$$\mathbf{A} = \begin{pmatrix} 2(1-\lambda^2) & \lambda^2 & & 0 \\ \lambda^2 & \ddots & \ddots & \\ & \ddots & \ddots & \lambda^2 \\ 0 & & \lambda^2 & 2(1-\lambda^2) \end{pmatrix}$$

with $\lambda = \alpha k/h$. This $(m-1)\times(m-1)$ structure accounts for the first set of boundary conditions. Note that i thus runs only over the $m-1$ subset, i.e. $i = 1, 2, \ldots, m-1$ here and in the following. The other boundary conditions are accounted for by initializing the first two time steps with

$$\begin{aligned} w_{i,j=0} &= f(x_i) \\ w_{i,j=1} &= (1-\lambda^2)f(x_i) + \frac{\lambda^2}{2}f(x_{i+1}) + \frac{\lambda^2}{2}f(x_{i-1}) + kg(x_i). \end{aligned}$$

Implement a Python function of the form `w = pdeHyperbolic(a, x, t, f, g)`, where `a` represents the PDE parameter α, `x` and `t` are the discretized spatial and time grids (use $j = 0, 1, \ldots, n$ steps for the time grid), and `f` and `g` are the functions defining the boundary conditions. This function should return `w[:,:]`, which is an array in the spatial and time coordinates (thus we aim to save the full solution at every time).

(b) Use you implementation to solve the following problems. Compare in the first case your numerical solution to the analytic one (thus use it to debug you implementation):

(i) Problem:

$$\begin{aligned}
\frac{\partial^2}{\partial t^2}u(x,t) - \frac{\partial^2}{\partial x^2}u(x,t) &= 0, && \text{for } 0 \le x \le 1 \text{ and } 0 \le t \le 1\\
u(0,t) = u(l,t) &= 0, && \text{for } t > 0\\
u(x,0) &= \sin(2\pi x),\\
\frac{\partial}{\partial t}u(x,0) &= 2\pi \sin(2\pi x) && \text{for } 0 \le x \le 1.
\end{aligned}$$

The corresponding analytic solution is $u(x,t) = \sin(2\pi x)(\cos(2\pi t) + \sin(2\pi t))$.

(ii) Problem:

$$\begin{aligned}
\frac{\partial^2}{\partial t^2}u(x,t) - \frac{\partial^2}{\partial x^2}u(x,t) &= 0, && \text{for } 0 \le x \le 1 \text{ and } 0 \le t \le 2\\
u(0,t) = u(l,t) &= 0, && \text{for } t > 0\\
u(x,0) &= \begin{cases} +1 & \text{for } x < 0.5\\ -1 & \text{for } x \ge 0.5 \end{cases},\\
\frac{\partial}{\partial t}u(x,0) &= 0 && \text{for } 0 \le x \le 1.
\end{aligned}$$

Use $m = 200$ and $n = 400$ to discretize the spatial and time grids, respectively.

(c) Animate your solutions! To this end you can use the following code:

```python
import matplotlib.pylab as plt
import matplotlib.animation as animation

fig = plt.figure(1)

def animate(f):   # is called to generate each frame of the animation
  plt.clf()       # clear the figure for each frame
  plt.ylim( ... ) # fix y limits
  plt.grid(True)  # ...
  plt.plot( ... ) # plot your (and analytic) solution here

frames = np.arange(1, np.size(t))  # t is the time grid here
myAnimation = animation.FuncAnimation(fig, animate, frames, interval = 1)
```

1.2 Parabolic PDEs: 1D Schrödinger Equation with Potential

Here we aim to solve the 1D Schrödinger equation (SEQ) for a particle exposed to some potential $V(x)$

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\Psi(x,t) + V(x)\Psi(x,t).$$

For simplicity we will set $\hbar = 1$ and $m = 0.5$ in the following. By using finite differences to approximate all involved derivatives we will first analyze the stationary solutions of the SEQ for a given potential. In the next step you will need to derive and apply the Crank-Nicolson approach for this PDE to study the dynamics of the SEQ.

(a) The stationary SEQ

$$H\Psi(x) = \frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\Psi(x) + V(x)\Psi(x) = E\Psi(x).$$

can be numerically studied by discretizing the spatial coordinate $x_i = ih$ and approximating the second derivative with finite differences. This results in an eigenvalue problem of the form $H\vec{\Psi} = E\vec{\Psi}$, where Ψ is now a vector in the discretized positions x_i. By diagoanlizing the H matrix we simultanously get both, the eigenenergies E_i and the corresponding eigenfunctions $\vec{\Psi}_i$. Derive the matrix representation of the Hamiltonian H and implement a simple Python function `E, Psi = SEQStat(x, V)` which takes the discretized grid x_i and the potential $V(x)$ as arguments and which returns all eigenvalues and eigenfunctions.

(b) Use your function to calculate and plot the first few eigenfunctions for $V(x) = 0.25x^2$. Add the potential to your plot. Use $-5 \leq x \leq +5$ discretized in 100 steps.

(c) Now we turn to the full dynamical problem, which can be numerically analyzed using the Crank-Nicolson approach which averages over forward and backward scattering finite differences for the time derivative. Your task is to derive the corresponding formulas for the 1D SEQ from above, implement it, and apply it to study the time-dynamics for the 1D SEQ. Applying the Crank-Nicolson approach to the 1D SEQ from above results in an equation system of the form

$$\mathbf{A}\vec{w}_{j+1} = \mathbf{B}\vec{w}_j$$

where $\vec{w}_j$ is the space- and time-discretized approximation to $\Psi(x_i, t_j)$ with j being the time-discretization index. $\mathbf{A}$ and $\mathbf{B}$ are $(m-1) \times (m-1)$ matrices of the form

$$\mathbf{A} = \begin{pmatrix} 1+\lambda_1+\lambda_2 V_1 & -\frac{\lambda_1}{2} & & 0 \\ -\frac{\lambda_1}{2} & \ddots & \ddots & \\ & \ddots & \ddots & -\frac{\lambda_1}{2} \\ 0 & & -\frac{\lambda_1}{2} & 1+\lambda_1+\lambda_2 V_{m-1} \end{pmatrix}$$

and

$$\mathbf{B} = \begin{pmatrix} 1-\lambda_1-\lambda_2 V_1 & +\frac{\lambda_1}{2} & & 0 \\ +\frac{\lambda_1}{2} & \ddots & \ddots & \\ & \ddots & \ddots & +\frac{\lambda_1}{2} \\ 0 & & +\frac{\lambda_1}{2} & 1-\lambda_1-\lambda_2 V_{m-1} \end{pmatrix}.$$

Derive this form and thereby the definitions of λ_i. Start with recapping the derivation of the Crank-Nicolson approach (see Wikipedia or *Numerical Analysis* book by Burden and Faires).

(d) The equation system from above can be used to calculate the time propagation by multiplying the whole expression from left with the inverse of $\mathbf{A}$. The only needed ingredients are the boundary conditions $w_{i=0,j} = w_{i=m,j} = 0$ for all j (not explicitly needed, just used for the derivation of $\mathbf{A}$ and $\mathbf{B}$) and the initial value $w_{i,j=0} = f(x_i)$. Implement the Crank-Nicolson approach for our 1D SEQ and apply it to the SEQ from above with $V(x) = 0.25x^2$ and $f(x) = \exp(-x^2)/2.5$. Use $-5 \leq x \leq +5$ with 100 steps and $0 \leq t \leq 50$ with 1000 steps. Again, animate your solution.

(e) (Optional:) What happens when your initial condition is set to one of the eigenfunctions you obtained from the static 1D SEQ?

(f) (Optional:) Change the potential to a staggered one of the form

$$V(x) = \begin{cases} +15 & -4.0 < x < -3.5,\ -2.5 < x < -2.0,\ -1.0 < x < -0.5, \\ +15 & +0.5 < x < +1.0,\ +2.0 < x < +2.5,\ +3.5 < x < +4.0, \\ 0 & \text{else} \end{cases}$$