Numerical methods for Partial Differential Equations

Maria Lewandowska

February 10 2021

Contents

The project aims to show different numerical methods for solving partial differential equations, including Sine-Gordon, Poisson and diffusion equations.

1 Partial differential equations

1. hyperbolic equations (Wave-type equations)

$$\partial_{tt}u - \partial_{xx}u + F(u,t,x) = 0$$

2. parabolic equations (Schroedinger, diffusion equation)

$$i\partial_{t}u + \partial_{xx}u + F(u,t,x) = 0$$

$$\partial_{t}u - \partial_{xx}u + F(u,t,x) = 0$$

3. elliptic (Laplace, Poisson, Helmholtz equation)

$$\partial_{tt}u + \partial_{xx}u + F(u,t,x) = 0$$

2 Poisson equation

$$\partial_{xx}\phi + \partial_{yy}\phi = f(x,y) \tag{1}$$

2.1 General scheme of solving Poisson equation (1):

1. discretize spatial dimensions:

$$x_p = p \cdot h \qquad p = 0, 1, 2, ...N$$
$$y_q = q \cdot h \qquad q = 0, 1, 2, ...N$$
$$\phi_{pq} = \phi(x_p, y_q)$$
$$f_{pq} = f(x_p, y_q)$$

   The lowest order $\mathcal{O}(h^2)$ disretization has the form:

$$\frac{\phi_{p+1,q} - 2\phi_{p,q} + \phi_{p-1,q}}{h^2} + \frac{\phi_{p,q+1} - 2\phi_{p,q} + \phi_{p,q-1}}{h^2} = f_{p,q}$$

$$\frac{\phi_{p+1,q} + \phi_{p-1,q} + \phi_{p,q+1} + \phi_{p,q-1} - 4\phi_{p,q}}{h^2} = f_{p,q}$$

   ϕ_{pq} and $f_{p,q}$ are matrices of size $N \times N$ which is problematic for solving an algebraic equation for ϕ_{pq}

2. vectorize ϕ_{pq} and f_{pq} using multiindex:

$$0 \leq I_{i,j} = iN + j < N^2$$

$$\phi_{I-N} + \phi_{I+N} + \phi_{I-1} + \phi_{I+1} - 4\phi_I = A_{IJ}x_J = h^2 \cdot f_I$$

$$A_{IJ} = \delta_{I-N,J} + \delta_{I+N,J} + \delta_{I-1,J} + \delta_{I+1,J} - 4\delta_{I,J}$$

3. vectorize boundary conditions:

$$\phi_{[0:N]} = \phi_{bottom} \qquad \phi_{[0:N(N-1):N]} = \phi_{left}$$
$$\phi_{[N(N-1):N^2]} = \phi_{top} \qquad \phi_{[(N-1):N^2:N]} = \phi_{right}$$

4. write a system of N^2 algebraic equations in a matrix form:

$$A\vec{\phi} = \vec{f}$$

   where:

   - A - matrix differentiation coefficients
   - $\vec{\phi}$ - vector of discretized coordinates
   - $\vec{f}$ - values of f evaluated at discrete points $f_I = f(x_p, y_q) \qquad I = p \cdot N + q$
   - $\vec{\phi}$ and A include boundary conditions

5. solve algebraic equation

$$A\vec{\phi} = \vec{f}$$

   - LU decomposition
   - np.linalg.solve()
   - sl.spsolve()

3 Poisson equation with periodic boundary conditions

Poisson equation in 2 dimensions:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \cos(3x + 4y) - \cos(5x - 2y) \tag{2}$$

with periodic boundary conditions:

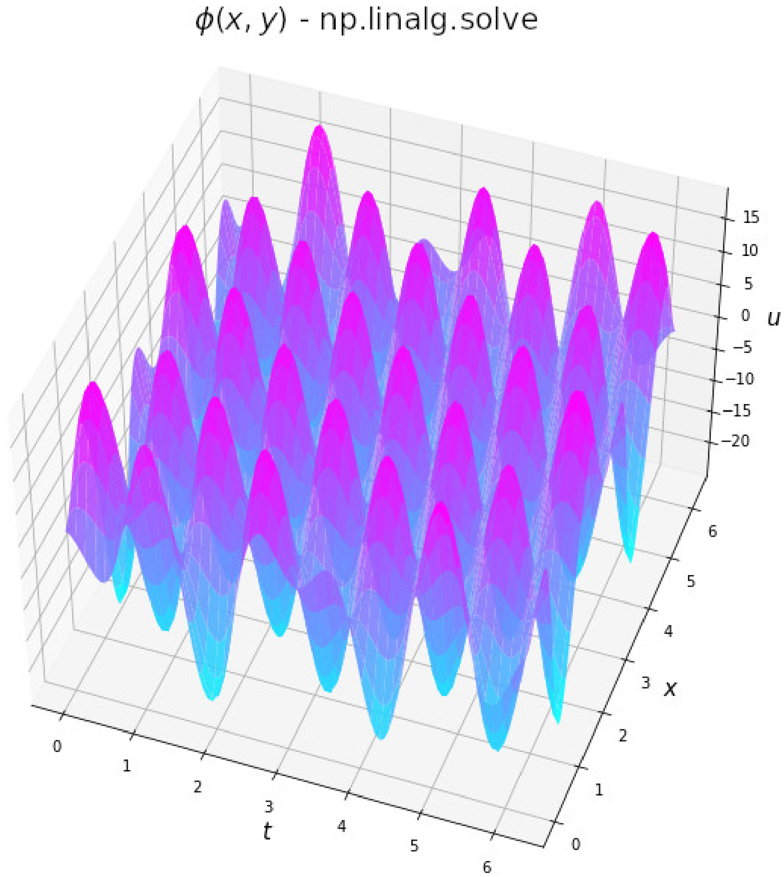
- $\phi(x, 0) = \phi(x, 2\pi)$
- $\phi(0, y) = \phi(2\pi, y)$

Analytical solution of (2):

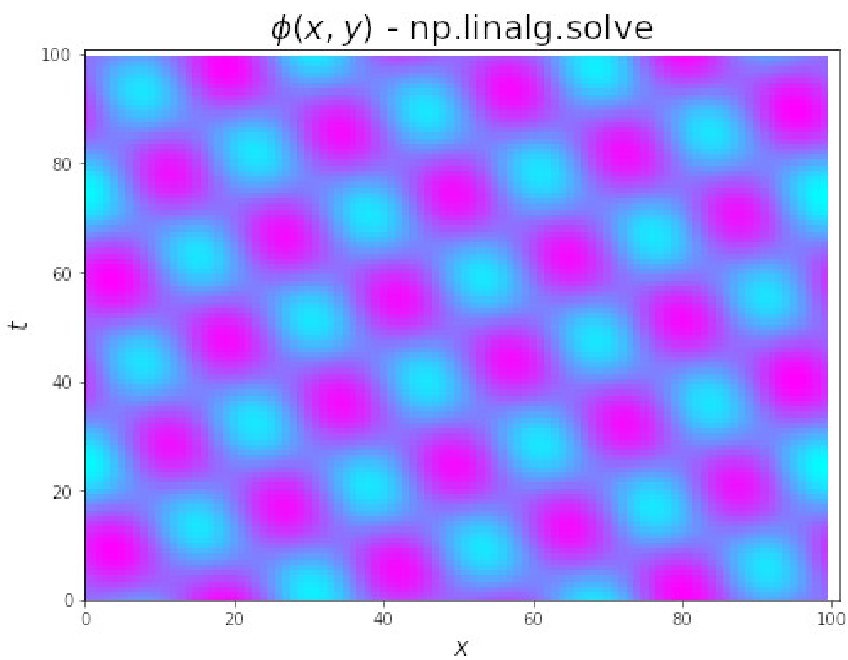
$$\phi(x, y) = -\frac{1}{25}\cos(3x + 4y) + \frac{1}{29}\cos(5x - 2y)$$

3.1 Implementation in Python

```
n = 100
xx = np.linspace(0, 2*PI, n)
yy = np.linspace(0, 2*PI, n)
dx = xx[1] - xx[0]
dy = yy[1] - yy[0]
x,y= np.meshgrid(xx,yy)

fmatrix = f(x,y)
fvector = fmatrix.flatten()

N = len(fvector)
D = np.diagflat( np.ones(N) * (-4))

for i in range(1,N-1):
    D[i][i] = -4
    D[i][i+1] = 1
    D[i][i-1] = 1
    D[i][(i+n)%N] = 1
    D[i][(i-n)%N] = 1

Periodic boundary conditions:
D[N-1][N-2] = 1
D[N-1][n-1] = 1
D[N-1][N-n-1] = 1
D[0][1]= 1
D[0][N-n]=1
D[0][n]=1

solver
Phi1vector = np.linalg.solve(D, fvector)
Phi2vector = sl.spsolve(D, fvector)
Phi1matrix = cp.copy(Phi1vector).reshape((n, n))
Phi2matrix = cp.copy(Phi2vector).reshape((n, n))
```

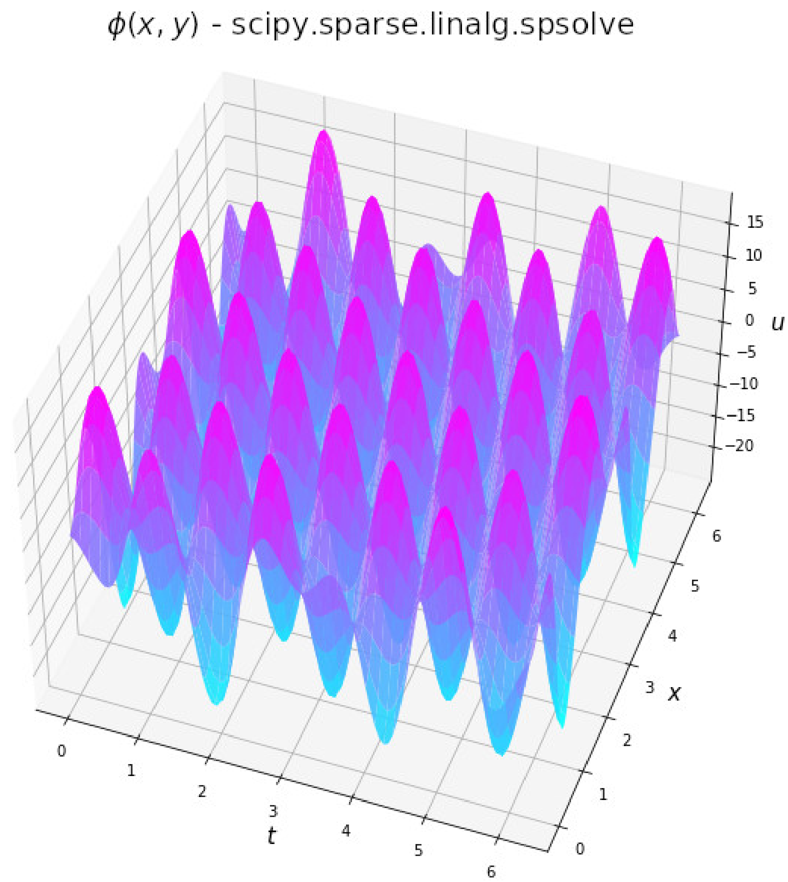


(a)

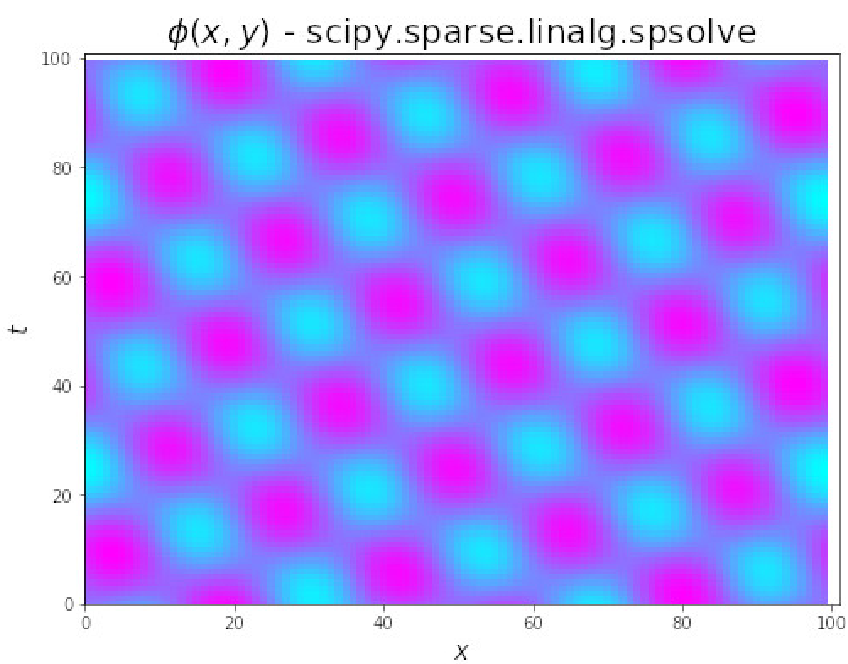


(b)

Figure 1: Solution of the Poisson equation (2) using np.linalg.solve()



(a)



(b)

Figure 2: Solution of the Poisson equation (2) using scipy.sparse.linalg.spsolve()

4 Discrete Fourier Transform for Poisson equation

For periodic or anty-periodic boundary conditions we may use Discrete Fourier Transform to solve Poisson equation

4.0.1 Discrete Fourier Transform and Inverse Discrete Fourier Transform:

DFT:

$$f_{pq} = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{2\pi ikp/n} e^{2\pi ilq/n} \cdot \tilde{f}_{kl}$$

IDFT:

$$\tilde{f}_{kl} = \sum_{p=0}^{n-1} \sum_{q=0}^{n-1} e^{-2\pi ikp/n} e^{-2\pi ilq/n} \cdot f_{pq}$$

4.0.2 Discretization scheme

- $x_p = \frac{2\pi p}{n}, p = 0, 1, 2, .., n-1$
- $y_q = \frac{2\pi q}{m}, q = 0, 1, 2, .., m-1$
- $\phi(x_p, y_q) = \phi_{p,q}$
- $f(x_p, y_q) = f_{p,q}$

Central-difference approximation to the second-derivative:

$$\frac{\phi_{p+1,q} - 2\phi_{p,q} + \phi_{p-1,q}}{h^2} + \frac{\phi_{p,q+1} - 2\phi_{p,q} + \phi_{p,q-1}}{h^2} = f_{p,q}$$

$$\phi_{p+1,q} + \phi_{p-1,q} + \phi_{p,q+1} + \phi_{p,q-1} - 4\phi_{p,q} = h^2 \cdot f_{p,q}$$

4.0.3 Solution in the Fourier space

We want to find $\phi_{p,q}$. We use DFT. In fourier space we dont have spatial derivatives. Let n = m for simplicity.

$$\phi_{pq} = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{2\pi ikp/n} \cdot e^{2\pi ilq/n} \cdot \tilde{\phi}_{kl}$$

$$f_{pq} = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{2\pi ikp/n} e^{2\pi ilq/n} \cdot \tilde{f}_{kl}$$

$p, q = 1, 2, ..., n-1$

We insert ϕ_{kl} and f_{kl} into the formula for central-difference approximation to the second-derivative:

$$\phi_{p+1,q} + \phi_{p-1,q} + \phi_{p,q+1} + \phi_{p,q-1} - 4\phi_{p,q} - h^2 \cdot f_{p,q} = 0$$

$$\frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{2\pi ikp/n} \cdot e^{2\pi ilq/n} \left[\tilde{\phi_{kl}} \left(e^{2\pi ik/n} + e^{-2\pi ik/n} + e^{2\pi il/n} + e^{-2\pi il/n} - 4 \right) - h^2 \cdot \tilde{f}_{kl} \right] = 0$$

We multiplicate both sides with $e^{-2\pi ik'p/n} e^{-2\pi il'q/n}$ and sum over p and q. From the orthonormality condition we get deltas.

$$\frac{1}{n^2} \sum_{p=0}^{n-1} \sum_{q=0}^{n-1} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{-2\pi ik'p/n} \cdot e^{-2\pi il'q/n} e^{2\pi ikp/n} \cdot e^{2\pi ilq/n} \left[\tilde{\phi}_{kl} \left(e^{2\pi ik/n} + e^{-2\pi ik/n} + e^{2\pi il/n} + e^{-2\pi il/n} - 4 \right) - h^2 \cdot \tilde{f_{kl}} \right] = 0$$

$$\frac{1}{n^2} \sum_{p=0}^{n-1} \sum_{q=0}^{n-1} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} e^{2\pi i(k-k')p/n} \cdot e^{2\pi i(l-l')q/n} \left[\tilde{\phi_{kl}} \left(e^{2\pi ik/n} + e^{-2\pi ik/n} + e^{2\pi il/n} + e^{-2\pi il/n} - 4 \right) - h^2 \cdot \tilde{f_{kl}} \right] = 0$$

$$\frac{1}{n^2}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}\delta_{k,k'}\delta_{l,l'}\left[\tilde{\phi}_{kl}\left(e^{2\pi ik/n}+e^{-2\pi ik/n}+e^{2\pi il/n}+e^{-2\pi il/n}-4\right)-h^2\cdot\tilde{f}_{kl}\right]=0$$

$$\tilde{\phi}_{k'l'}\left[2\cos\left(2\pi\frac{k'}{n}\right)+2\cos\left(2\pi\frac{l'}{n}\right)-4\right]-h^2\cdot\tilde{f}_{k'l'}=0$$

Finally, we get a nice formula for $\tilde{\phi}_{k'l'}$

$$\tilde{\phi}_{k'l'}=\frac{1}{2}\cdot\frac{h^2\cdot\tilde{f}_{k'l'}}{\cos\left(2\pi\frac{k'}{n}\right)+\cos\left(2\pi\frac{l'}{n}\right)-2}$$

4.0.4 Solution in the real space

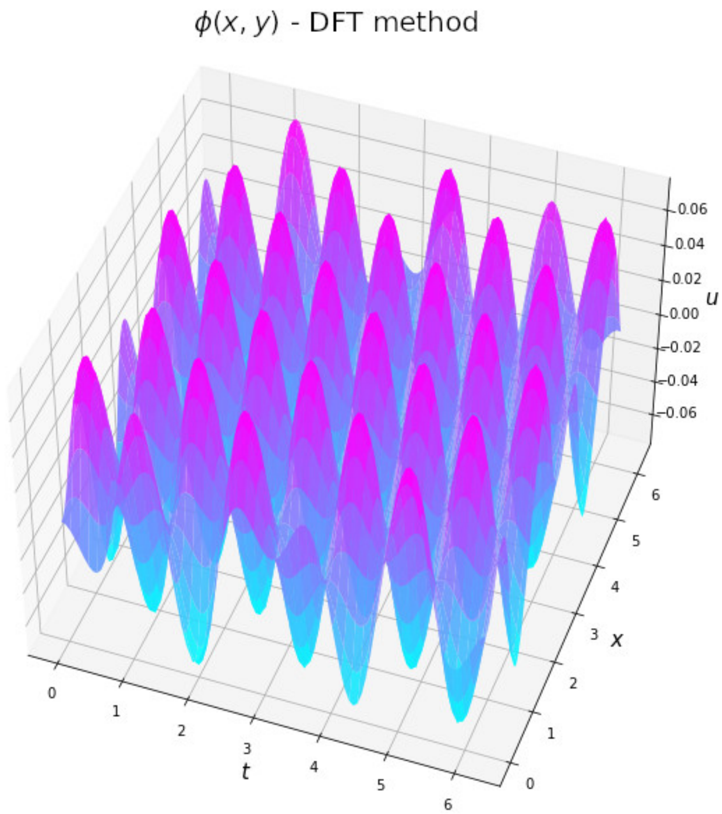
Using IDFT we get ϕ_{kl}

$$\phi_{pq}=\frac{1}{n^2}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}e^{2\pi ikp/n}\cdot e^{2\pi ilq/n}\cdot\tilde{\phi}_{kl}$$

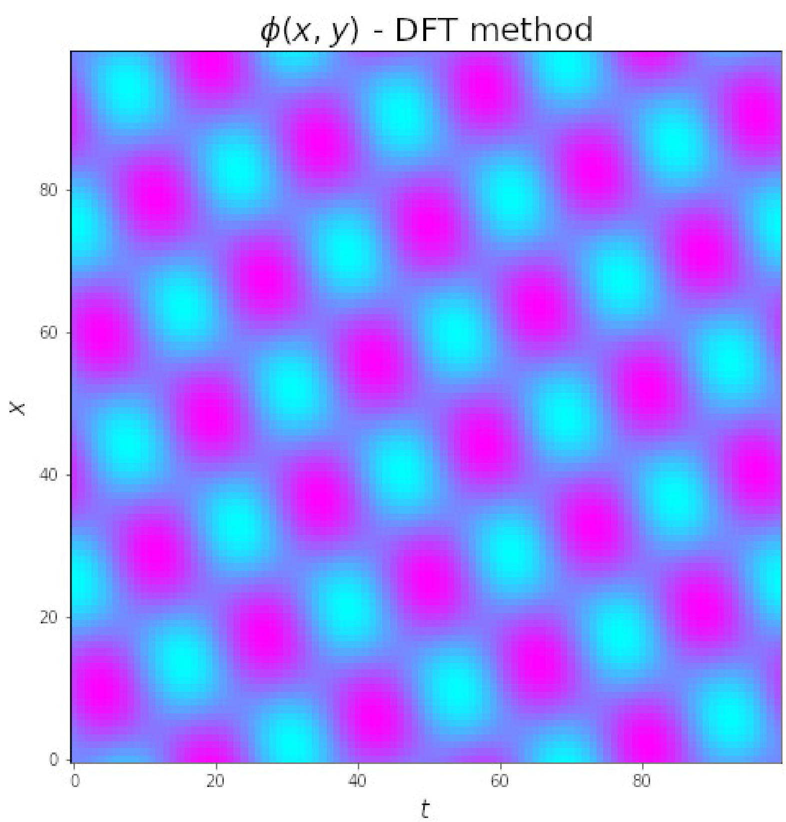
4.1 Implementation in Python

```
def DFT_1(f,n):
    f = f.astype(complex)
    f_tilde = np.zeros_like(f).astype(complex)
    for k in range(n):
        for l in range(n):
            for p in range(n):
                for q in range(n):
                    f_tilde[k][l] += f[p][q] * np.exp(-2*PI*1j*k*p/n) * np.exp(-2*PI*1j*l*q/n)
    return f_tilde



def IDFT_1(f_tilde,n):
    f_tilde = f_tilde.astype(complex)
    f = np.zeros_like(f_tilde).astype(complex)
    for p in range(n):
        for q in range(n):
            for k in range(n):
                for l in range(n):
                    f[p][q] += f_tilde[k][l] * np.exp(2*PI*1j*k*p/n) * np.exp(2*PI*1j*l*q/n)
    return f.real/(n*n)


def get_coeffs(f_tilde, n, h):
    f_tilde = f_tilde.astype(complex)
    g_tilde = np.zeros_like(f_tilde).astype(complex)
    for k in range(n):
        for l in range(n):
            g_tilde[k][l] = 0.5 * h * h * f_tilde[k][l]/ (np.cos( 2*PI*k/n) + np.cos( 2*PI*l/n)-2)
    g_tilde[0][0] = 0
    return g_tilde


def Poisson_DFT(f,n):
    xp_ = np.linspace(0,n-1, n)* 2* PI /n
    yq_ = np.linspace(0,n-1, n)* 2* PI /n
    xp,yq = np.meshgrid(xp_,yq_)
```

```
    h = xp_[1] - xp_[0]

    f_pq = f(xp,yq)
    f_tilde_kl = DFT_1(f_pq, n)

    g_tilde = get_coeffs(f_tilde_kl, n, h)
    g = IDFT_1(g_tilde, n)
    return g
```



(a)



(b)

Figure 3: Solution of Poisson equation using DFT method

5 Hyperbolic equations

$$\partial_{tt}u(t,x) - \partial_{xx}u(t,x) + F(u,t,x) = 0 \tag{3}$$

5.1 General scheme of solving hyperbolic equations (3):

1. discretize spatial dimension:
$$x_n = nh \qquad n = 0,1,2,...N-1$$
$$u_n(t) = u(t,x_n) \qquad n = 0,1,2,...N-1$$

2. write a system of N second order time differential equations:
$$\partial_{xx}u_n = \frac{u_{n-1} - 2u_n + u_{n+1}}{h^2}$$
$$\partial_{tt}u_n = D_{nm}u_m - F_n$$
$$D_{n,m} = \frac{1}{dx^2} \cdot (\delta_{n-1,m} - 2\delta_{n,m} + \delta_{n+1,m}), \qquad n,m = 0,1,2,3,...,N-1$$

3. use some time-stepping method to solve the system for given initial conditions $u(x,0), \partial_t u(x,0)$ and boundary conditions
   - simplectic method
   - Runge-Kutta method

5.2 Time-stepping methods - theory:

Let:

- $n,m = 0,1,2,3,..Nx$ - indexing of spatial coordinates
- $i,j = 0,1,2,3,..Nt$ - indexing of temporal coordinates

5.2.1 simplectic method:

$$\partial_t u^i = v^i$$
$$\partial_{tt}u^i = \partial_t v^i = G(u^i)$$

where:
$$G(u^i) = G(u^i_n) = D_{n,m}u_m - F_n$$

For every iteration we perform a multi-step:
$$u^{i+1/2} = u^i + \frac{1}{2}v^i dt = k^i$$
$$v^{i+1} = v^i + G\left(u_{i+1/2}\right)dt$$
$$u^{i+1} = u^{i+1/2} + \frac{1}{2}v^i dt$$

5.2.2 Runge–Kutta–Nyström methods:

$$u'' = F(x, u, u')$$

For every iteration we perform a multi-step:

$$u_{n+1} = u_n + hu'_n + h^2 \sum_{i=0}^{m} \bar{b}_i k'_i$$

$$u'_{n+1} = u'_n + h \sum_{i=0}^{m} b_i k'_i$$

$$k'_i = F(x_n + c_i h, u_n + c_i h u'_n + h^2 \sum_{j=1}^{m} \bar{a}_{ij} k'_j)$$

m = number of stages
a_{ij} , b_i, c_i parameters taken from the Butcher tableau

c_i	0			a_{ij}
	$\frac{1}{5}$	$\frac{1}{50}$		
	$\frac{2}{3}$	-$\frac{1}{27}$	$\frac{7}{27}$	
	1	$\frac{3}{10}$	-$\frac{2}{35}$	$\frac{9}{35}$
	b_i	$\frac{14}{336}$	$\frac{100}{366}$	0
	d_i	$\frac{14}{336}$	$\frac{125}{366}$	$\frac{35}{336}$

5.3 Implementation in Python

```python
def simplectic_1(Nt, Nx, u, v, dx,dt, D):
    for i in range(0,Nt):
        tmp = np.empty(Nx+1)
        tmp[:] = u[i, :] + 0.5 * v[i, :] * dt
        v[i+1,:] = v[i, :] - np.sin(tmp)* dt
        v[i+1,:] += dt/(dx*dx) * np.matmul(D, tmp)
        u[i+1,:] = tmp + 0.5 * v[i, :]* dt
        u[i+1:,0] = np.sin(omega1 * (i+1)*dt)
        u[i+1:,-1] = np.sin(omega2 * (i+1)*dt)
    return u,v


def simplectic_2(Nt, Nx, u, v, dx, dt):
    for i in range(0,Nt):
        tmp = np.zeros(Nx+1)
        tmp[:] = u[i,:] + 0.5 * v[i, :] * dt
        v[i+1,:] = v[i, :] - np.sin(tmp)* dt
        v[i+1, 1:-1] += dt/(dx*dx) * (tmp[:-2] - 2*tmp[1:-1] + tmp[2:])
        u[i+1,1:-1] = (tmp + 0.5 * v[i, :]* dt)[1:-1]
    return u,v


def simplectic_3(Nt, Nx, u, v, dx,dt):
    for i in range(0,Nt):
        tmp = np.zeros(Nx+1)
        tmp[:] = u[i,:] + 0.5 * v[i, :] * dt
        v[i+1,:] = v[i, :] - np.sin(tmp)* dt
        v[i+1, 1:-1] += dt/(dx*dx) * np.diff(tmp[:], 2)
        u[i+1,1:-1] = (tmp + 0.5 * v[i, :]* dt)[1:-1]
    return u,v
```

```python
def Runge_Kutta_Nystrom_1(Nt, Nx, u, v, dx, dt, a,b,d,c):

    for i in range(0,Nt):
        k0 = np.zeros(Nx+1)
        k1 = np.zeros(Nx+1)
        k2 = np.zeros(Nx+1)
        k3 = np.zeros(Nx+1)

        k0[1:-1] = f(u[i, :], dx)
        k1[1:-1] = f(u[i, :] + c[1]*dt*v[i, :] + dt*dt*a[1][0]*k0, dx)
        k2[1:-1] = f(u[i, :] + c[2]*dt*v[i, :] + dt*dt*a[2][0]*k0 + dt*dt*a[2][1]*k1, dx)
        k3[1:-1] = f(u[i, :] + c[3]*dt*v[i, :] + dt*dt*a[3][0]*k0 + dt*dt*a[3][1]*k1 +
            dt*dt*a[3][2]*k2, dx)
        u[i+1,1:-1] = u[i, 1:-1] + dt * v[i, 1:-1] + dt*dt* (b[0]*k0 + b[1]* k1 + b[2]*k2 + b[3]*k3
            )[1:-1]
        v[i+1,:] = v[i, :]
        v[i+1, :] += dt* (d[0]*k0 + d[1]* k1 + d[2]*k2 + d[3]*k3 )
    return u,v
```

6 Sine-Gordon equation:

Sine-Gordon equation is a nonlinear hyperbolic partial differential equation in 1 + 1 dimensions.

$$\partial_{tt}u = \partial_{xx}u - \sin(u) \tag{4}$$

Discretized Sine-Gordon equation takes the following form:

$$\partial_{tt}u_n = \frac{u_{n-1} - 2u_n + u_{n+1}}{h^2} - \sin(u_n) = D_{n,m}u_m - sin(u_n) \tag{5}$$

where: $D_{n,m} = \delta_{n-1,m} - 2\delta_{n,m} + \delta_{n+1,m}, \qquad n, m = 0, 1, 2, 3, ..., Nx$

One of realizations of discretized Sine-Gordon equation is a row of pendulums hanging from a rod that are coupled by torsion springs. It this case, $u_n(t)$ and $u'_n(t)$ correspond to the position and velocity of n-th pendulum. In order to numerically solve (4) we need to impose some initial conditions ($u_n(0)$, $u'_n(0)$) and boundary conditions ($u_0(t)$, $u_{Nx}(t)$). The initial and boundary conditions determinate time evolution of the system.

6.1 Models under investigation:

6.1.1 model 1

- initial conditions:

$$u_n(0) = 0 \qquad \forall n = 0, 1, 2, ...Nx$$

$$\partial_t u_n(0) = \begin{cases} \omega & for \quad n = 0 \\ \omega & for \quad n = Nx \\ 0 & for \quad 0 < n < Nx \end{cases}$$

- boundary conditions:

$$u(0, t) = \sin(\omega t)$$

$$u(L, t) = \sin(\omega t)$$

6.1.2 model 2

- initial conditions:

$$u_n(0) = 0 \qquad \forall n = 0, 1, 2, ...Nx$$

$$\partial_t u_n(0) = \begin{cases} \omega & for \quad n = 0 \\ -\omega & for \quad n = Nx \\ 0 & for \quad 0 < n < Nx \end{cases}$$

- boundary conditions:

$$u(0, t) = \sin(\omega t)$$

$$u(L, t) = -\sin(\omega t)$$

6.1.3 model 3

- initial conditions:

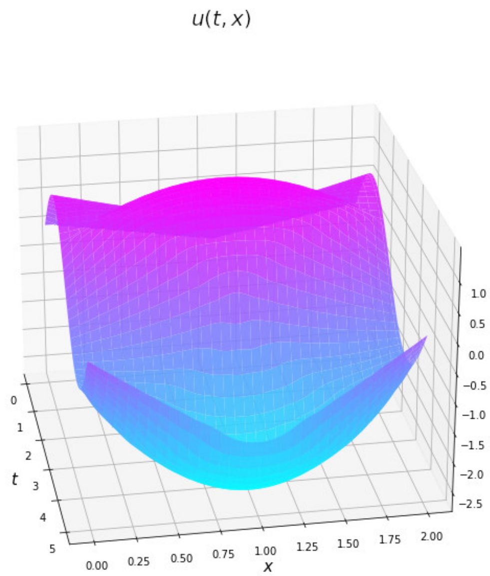
$$u_n(0) = 0 \qquad \forall n = 0, 1, 2, ...Nx$$

$$\partial_t u_n(0) = \begin{cases} \omega_1 & for \quad n = 0 \\ \omega_2 & for \quad n = Nx \\ 0 & for \quad 0 < n < Nx \end{cases}$$

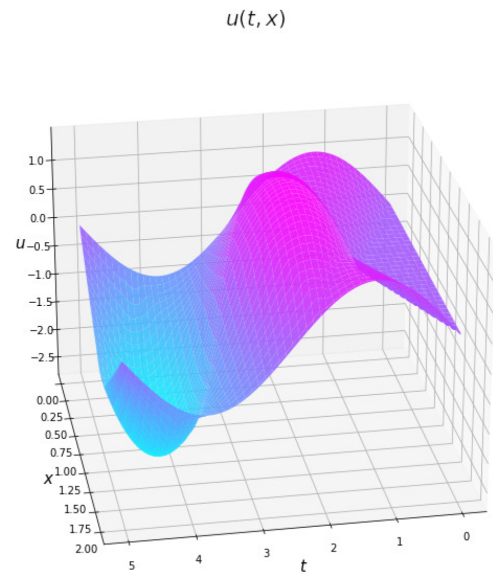
- boundary conditions:

$$u(0, t) = \sin(\omega_1 t)$$

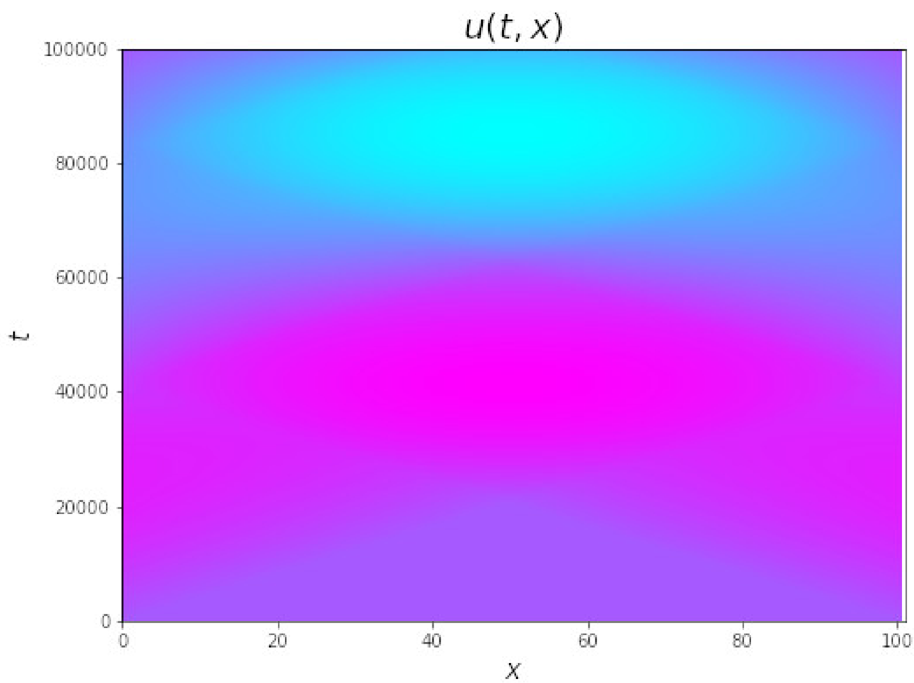
$$u(L, t) = \sin(\omega_2 t)$$



(a)

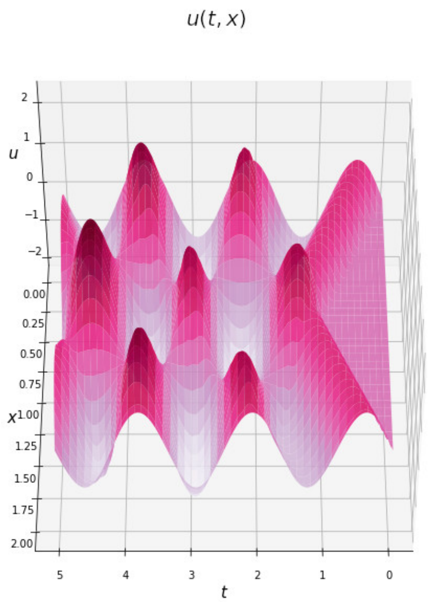


(b)

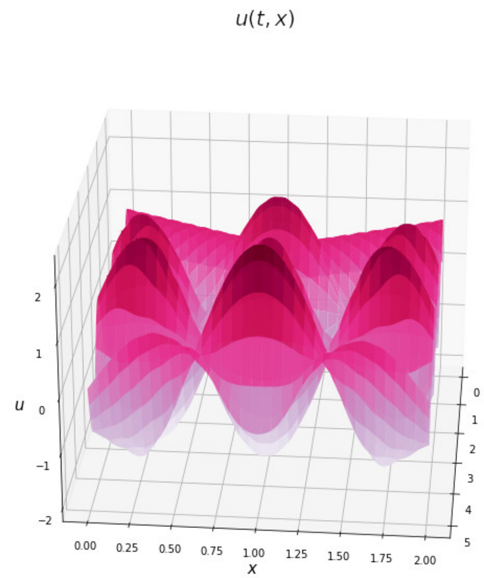


(c)

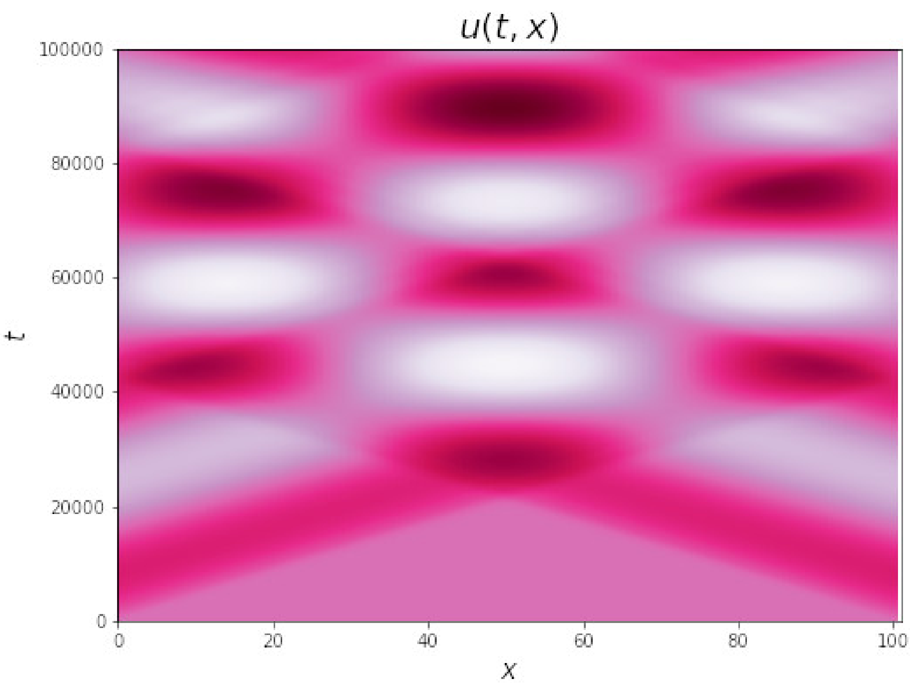
Figure 4: Model 1: Solution of Sine Gordon equation u(x,t) for $T = 5$, $L = 2$, $\omega = 2\pi/T$



(a)

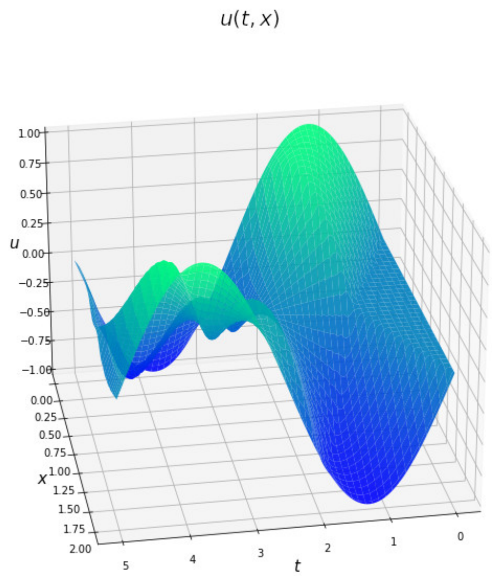


(b)

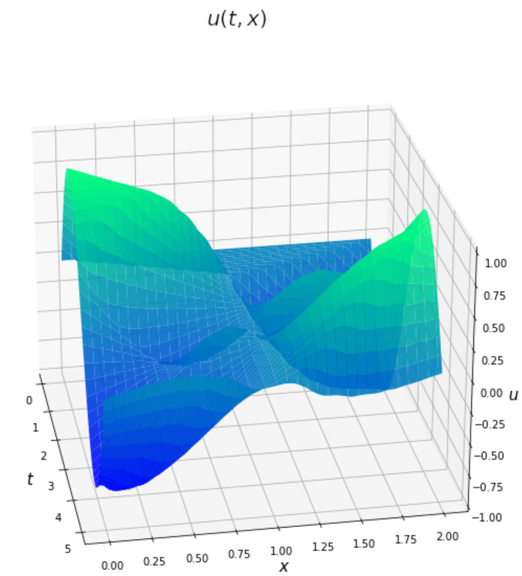


(c)

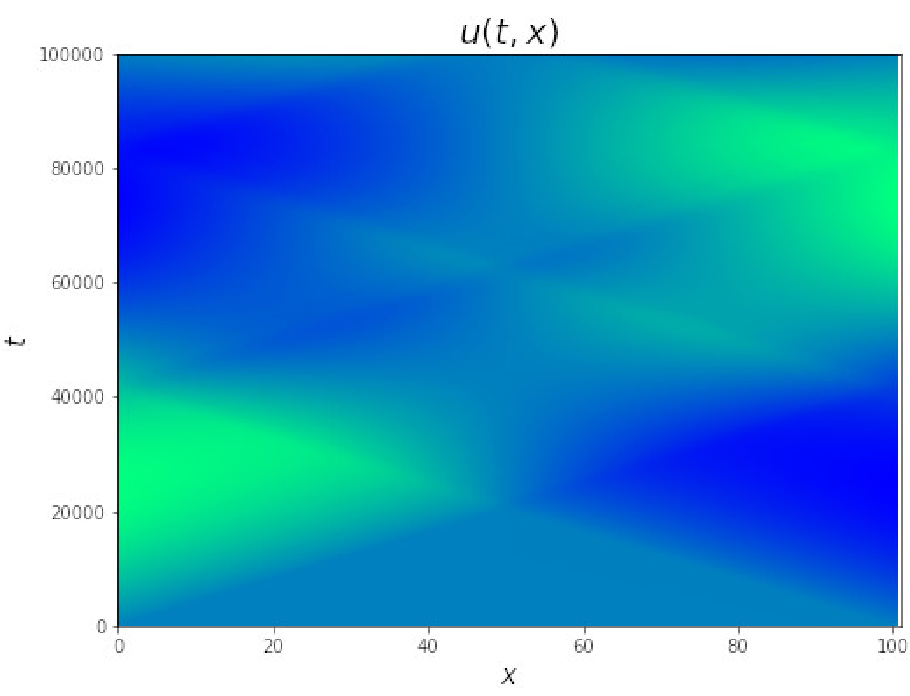
Figure 5: Model 1: Solution of Sine-Gordon equation u(x,t) for $T = 5$, $L = 2$, $\omega = 6\pi/T$



(a)



(b)



(c)

Figure 6: Model 2: Solution of Sine-Gordon equation u(x,t) for $T = 5$, $L = 2$, $\omega = 2\pi/T$

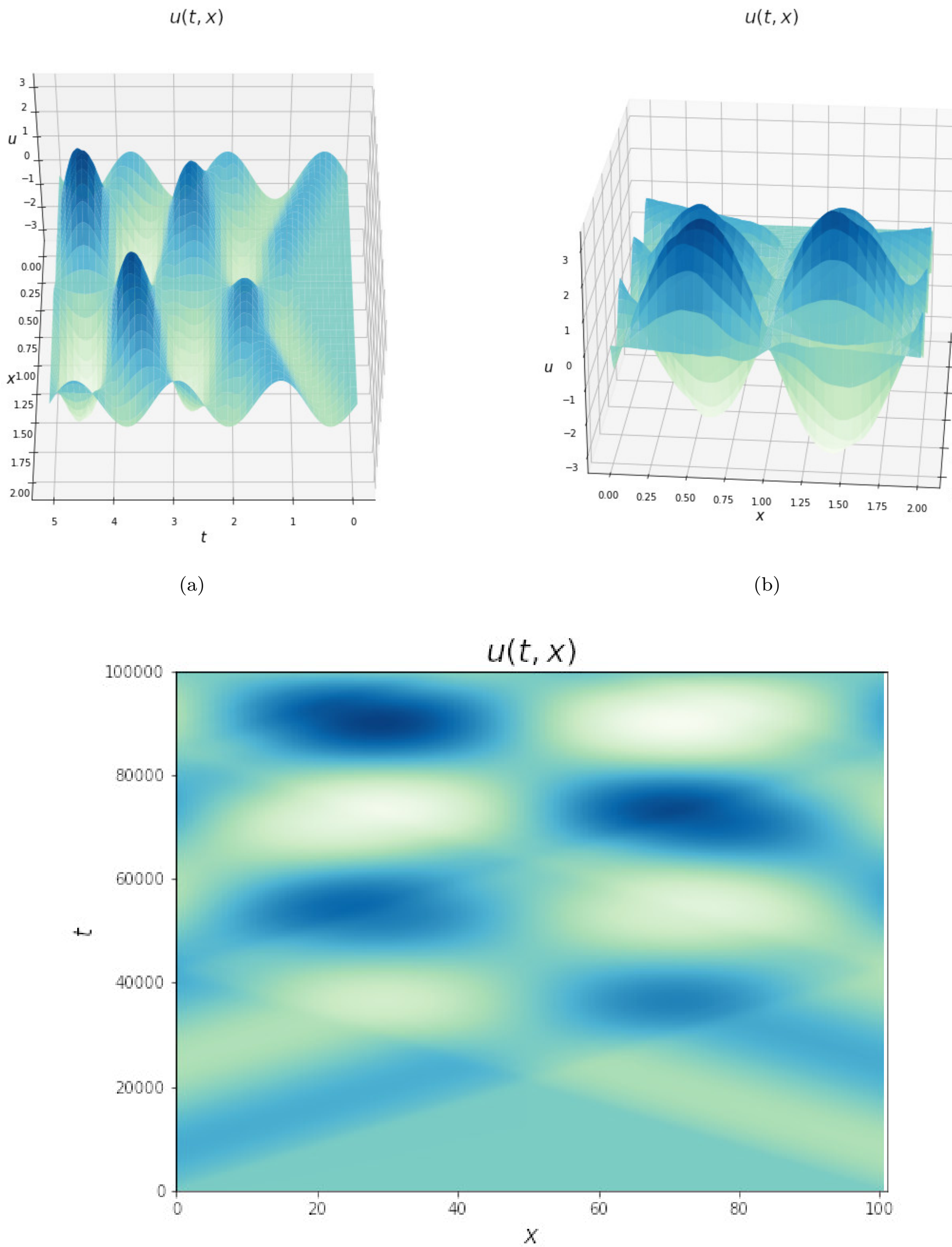
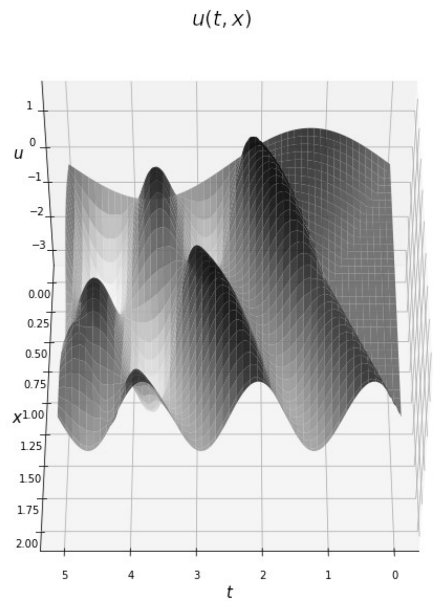
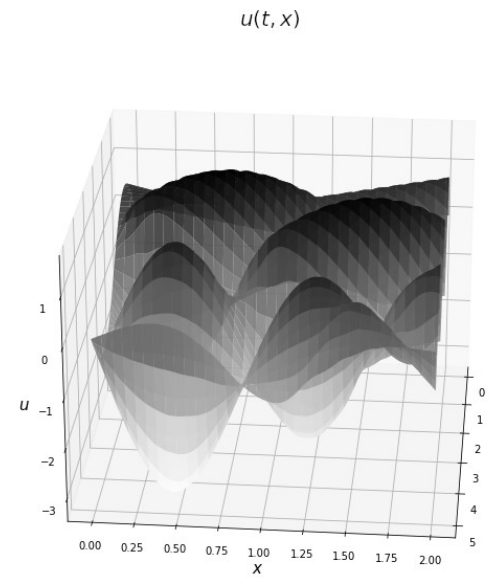


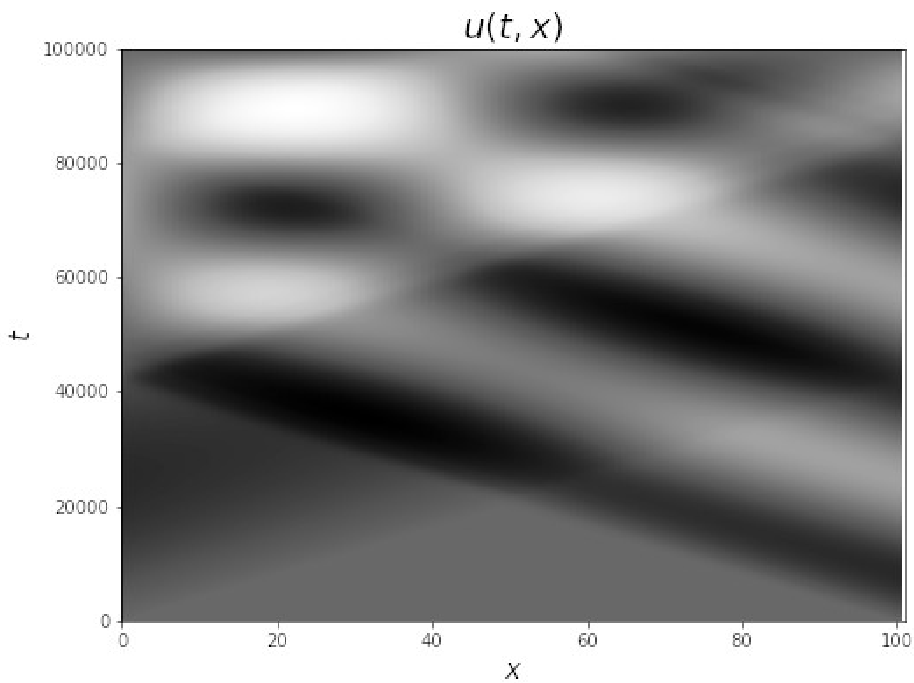
Figure 7: Model 2: Solution of Sine-Gordon equation u(x,t) for $T = 5$, $L = 2$, $\omega = 6\pi/T$



(a)



(b)



(c)

Figure 8: Model 2: Solution of Sine-Gordon equation u(x,t) for $T = 5$, $L = 2$, $\omega_1 = 2\pi/T$, $\omega_2 = 6\pi/T$

6.1.4 Time measurements

Measurements of how much time takes 10x evaluation of each function

	model 1	model 3
simplectic 1	user 8.84 s, sys: 3.73 ms	user 8.73 s, sys: 0 ns
simplectic 2	user 3.08 s, sys: 173 ms	user 4.02 s, sys: 36 ms
simplectic 3	user 3.47 s, sys: 64.3 ms	user 3.58 s, sys: 116 ms
Runge_Kutta_Nyström	user 11.5 s, sys: 10.2 ms	user 10.7 s, sys: 27.7 ms
Runge_Kutta_Nyström + jit	user 866 ms, sys: 15.9 ms	user 835 ms, sys: 0 ns,

7 Diffusion equation

Diffusion equation is an example of a parabolic equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{6}$$

with initial conditions:

- $u(x,0) = \sin \pi x L$

and boundary conditions:

- u(0,t) = 0
- u(L,t) = 0

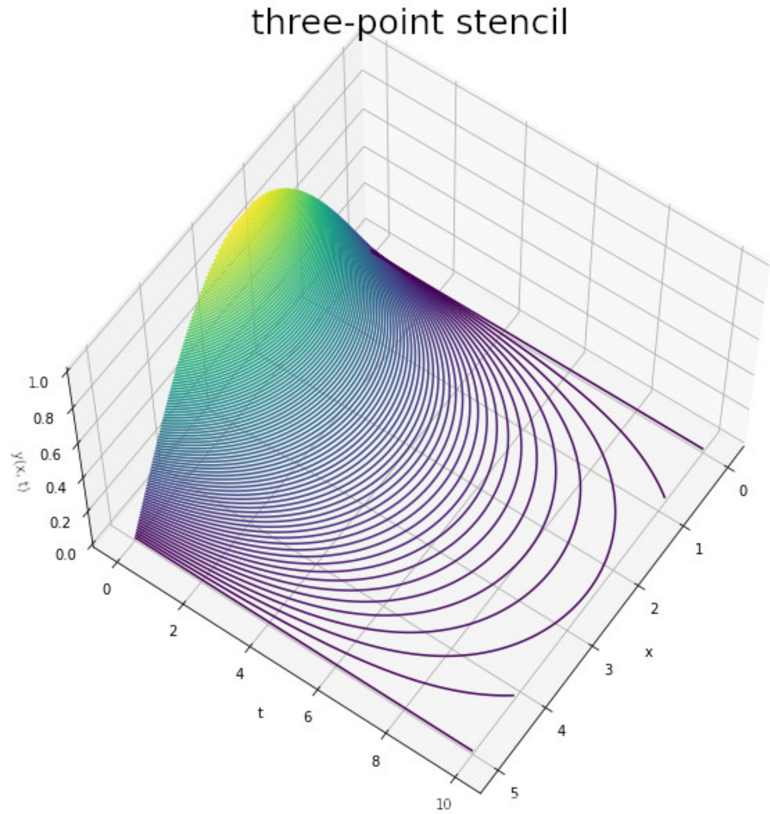
7.1 Scheme of solving diffusion equation (6):

1. discretize spatial dimension:
$$x_n = n\Delta x \qquad n = 0,1,2,...N_x$$
$$u_n(t) = u(t, x_n) \qquad n = 0,1,2,...N_x$$
2. use centered difference formulas for five-point or three point stencils approximating second derivative:
    - five-point strencil:
$$f''(x) \approx \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2}$$
$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u}{\partial x^2} = \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2}$$
$$\frac{u_i^{n+1} - u_i^n}{t^{n+1} - t^n} = \frac{-u_{i+2}^n + 16 \cdot u_{i+1}^n - 30 \cdot u_i^n + 16 \cdot u_{i-1}^n - u_{n-2}}{12(x_{i+1} - x_i)^2}$$
    - three-point strencil:
$$\frac{u_i^{n+1} - u_i^n}{t^{n+1} - t^n} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(x_{i+1} - x_i)^2}$$
3. using a substitution:
$$\Delta x = x_{i+1} - x_i, \qquad i \in (0, N_x)$$
$$\Delta t = t^{n+1} - t^n \qquad n \in (0, N_t)$$

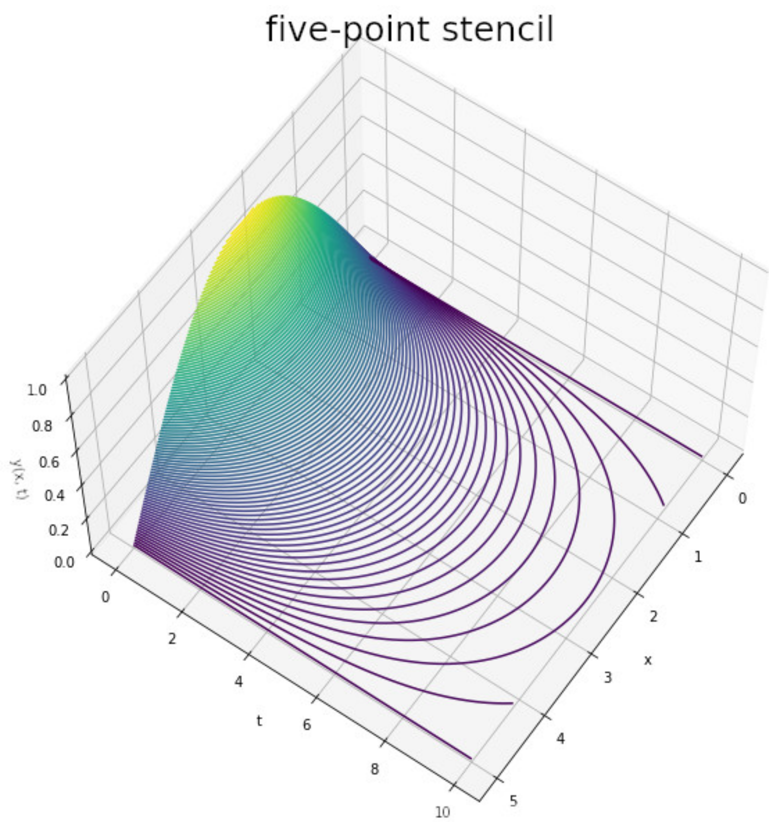
    rewrite (6) to the form:
$$u_i^{n+1} = u_i^n + \frac{\Delta t}{12\Delta x^2}\left[-u_{i+2}^n + 16 \cdot u_{i+1}^n - 30 \cdot u_i^n + 16 \cdot u_{i-1}^n - u_{n-2}\right]$$
4. use the initial and the boundary conditions:
    - $\forall n : u_0^n = 0$
    - $\forall n : u_{Nx}^n = 0$
    - $\forall i : u_i^0 = \sin\left(\frac{\pi x_i}{L}\right)$
5. use the symmetry properties of $\sin\left(\frac{\pi x_i}{L}\right)$:
    - $\forall n : u_{-1}^n = -u_1^n$
    - $\forall n : u_{L+1}^n = -u_L^n$
6. solve (6) as an equation for initial value problem

7.2 Implementation in Python

```
def thee_point_strencil(y, Nx, Nt, xx, tt):
    F = dt/(dx**2)
    for i in range(1,Nt+1):
        for j in range(1,Nx):
            y[i][j] = y[i-1][j] + F*(y[i-1][j-1] - 2*y[i-1][j] + y[i-1][j+1])
return y


def five_point_strencil(y, Nx, Nt, xx, tt):
    F = dt/(12*dx**2)
    for n in range(1,Nt+1):
        for i in range(2,Nx-1):
            y[n][i] = y[n-1][i] + F*(-y[n-1][i+2] + 16*y[n-1][i+1] - 30*y[n-1][i] +
                16*y[n-1][i-1]-y[n-1][i-2])
        y[n][1] = y[n-1][1] + F*(-y[n-1][3] + 16*y[n-1][2] - 30*y[n-1][1] + 16*y[n-1][0]+y[n-1][1])
        y[n][Nx-1] = y[n-1][Nx-1] + F*(y[n-1][Nx-1] + 16*y[n-1][Nx] - 30*y[n-1][Nx-1] +
            16*y[n-1][Nx-2] -y[n-1][Nx-3])
    return y
```



(a)



(b)

Figure 9: Solution of the diffusion equation (6). Three-point [top] and five-point [bottom] strencil.