Основные формулы

- **Волновое уравнение (задача Коши)** для функции $E(t,z)$ (на бесконечной прямой) записывается как $$\frac{\partial^2 E}{\partial t^2} = c^2 \frac{\partial^2 E}{\partial z^2},\qquad E(0,z)=\varphi(z),\;\;\frac{\partial E}{\partial t}(0,z)=\psi(z),$$ где c – скорость волны [1]. В простейшем случае $\psi(z)\equiv0$.

- **Формула Даламбера** (аналитическое решение) при $\psi=0$ имеет вид $$E(t,z) = \frac{\varphi(z-ct)+\varphi(z+ct)}{2}\,$$ (то есть начальный профиль распадается на два волновых фронта) [2].

- **Начально–граничная задача** на конечном отрезке $-L<z<L$ с нулевыми краевыми условиями: $$\frac{\partial^2 E}{\partial t^2} = c^2 \frac{\partial^2 E}{\partial z^2},\quad -L<z<L, \;t>0,$$ $$E(t,-L)=0,\;E(t,L)=0,\quad E(0,z)=\varphi(z),\;\frac{\partial E}{\partial t}(0,z)=0.$$

- **Явная разностная схема** второго порядка по времени и пространству (вариант A, центральные разности) для внутренних узлов j: $$E_j^{n+1} = 2E_j^n - E_j^{n-1} + \left(\frac{c\,\tau}{h}\right)^2 \bigl(E_{j+1}^n - 2E_j^n + E_{j-1}^n\bigr),$$ где τ и h – временной и пространственный шаги соответственно. При $\tau=\frac{h}{c}$ такая схема устойчива (условие Куранта–Фридрихса–Леви).

Структура Python-классов

- **Класс `WaveEquation` (или `WaveSolver`).** Конструктор принимает параметры задачи: полуширину области L, скорость c, число пространственных точек N_x, число временных шагов N_t (обязательный параметр), начальную функцию $\varphi(z)$ (например, в виде Python-функции) и, при необходимости, начальную скорость $\psi(z)$.
- **Метод** `solve_numeric()` внутри класса строит численное решение явной разностной схемой. Он выделяет массив формата `(N_t+1)×N_x`, итерирует по времени по формуле выше, применяя граничные условия (например, $E=0$ на краях).
- **Метод** `solve_analytic(t)` строит аналитическое решение по формуле Даламбера для заданного времени t. Он вычисляет $E(t,z)=\frac12[\varphi(z-ct)+\varphi(z+ct)]$, используя интерполяцию значений начального профиля на сетке.

- **Поля класса**: массивы координат по z, параметры шагов Δz, Δt, Курантово число $r=c\,\Delta t/\Delta z$, и собственно массив численного решения.

- **Визуализация**. При использовании библиотеки matplotlib можно в одном графике построить численное решение (например, последнего времени $E_j^{N_t}$) и аналитическое $E(z,t)$ (штриховой линией). Легенда обозначает “Численное” и “Аналитическое” решения, оси – z и E. Пример графика приведён ниже.

```python
class WaveEquation:
    def __init__(self, L, c, Nx, Nt, phi_func, psi_func=None):
        self.L = L
        self.c = c
        self.Nx = Nx
        self.Nt = Nt
        self.x = np.linspace(-L, L, Nx)
        self.dx = self.x[1] - self.x[0]
        self.phi = phi_func(self.x)
        self.psi = np.zeros_like(self.x) if psi_func is None else psi_func(self.x)
        self.dt = self.dx / self.c       # выбираем dt для устойчивости
        self.r = self.c * self.dt / self.dx
        self.E = np.zeros((Nt+1, Nx))

    def solve_numeric(self):
        # Начальные условия
        self.E[0, :] = self.phi
        self.E[1, 1:-1] = self.E[0, 1:-1] + 0.5 * self.r**2 * (
            self.E[0, 2:] - 2*self.E[0, 1:-1] + self.E[0, :-2]
        )
        # Краевые условия
        self.E[1, 0] = 0; self.E[1, -1] = 0
        # Шаг по времени
        for n in range(1, self.Nt):
            self.E[n+1,1:-1] = (2*self.E[n,1:-1] - self.E[n-1,1:-1] +
                                self.r**2 * (self.E[n,2:] - 2*self.E[n,1:-1] + self.E[n,:-2]))
            self.E[n+1,0] = 0; self.E[n+1,-1] = 0
        return self.E

    def solve_analytic(self, t):
        # Аналитическое решение по Даламберу
        xp = self.x + self.c*t
        xm = self.x - self.c*t
        F = self.phi
        E_a = 0.5*(np.interp(xp, self.x, F, left=0, right=0) +
                   np.interp(xm, self.x, F, left=0, right=0))
        return E_a

# Пример использования (вариант A)
import matplotlib.pyplot as plt
phi = lambda z: np.exp(-50*z**2)  # начальный профиль (например, гаусс)
solver = WaveEquation(L=1.0, c=1.0, Nx=201, Nt=200, phi_func=phi)
E_num = solver.solve_numeric()
t_final = solver.dt * solver.Nt
E_an = solver.solve_analytic(t_final)

plt.plot(solver.x, E_num[-1,:], label='Численное')
```

```
plt.plot(solver.x, E_an, '--', label='Аналитическое')
plt.legend(); plt.xlabel('z'); plt.ylabel('E');
plt.title(f'Волновое поле в момент t={t_final:.2f}')
plt.show()
```

[2] [1] Этот код решает задачу A численно и строит график сравнения. На графике видно, как численное решение (сплошная линия) совпадает с аналитическим (штриховая) до момента отражения от границ. Источники формул: волновое уравнение [1] и формула Даламбера [2].

---

[1] [2] **main.dvi**
https://www.iist.ac.in/sites/default/files/people/IN08026/dAlembert.pdf