

Методические материалы: Разностные схемы для ОДУ колебаний

С. Лемешевский¹

Институт математики НАН Беларуси¹

Mar 13, 2017

Простейшая модель колебательного процесса

$$u'' + \omega^2 u = 0, \quad t \in (0, T], \quad (1)$$

$$u(0) = U, \quad u'(0) = 0. \quad (2)$$

Точное решение

$$u(t) = U \cos \omega t, \quad (3)$$

описывает колебания с постоянной амплитудой U и угловой частотой ω . Соответствующий период колебаний равен $P = 2\pi/\omega$.

Разностная схема

- Введем равномерную сетку по переменной t с шагом τ :

$$\omega_\tau = \{t_n = n\tau, \quad n = 0, 1, \dots, N\}.$$

- Простейшая разностная схема

$$\frac{y^{n+1} - 2y^n + y^{n-1}}{\tau^2} = -\omega^2 y^n. \quad (4)$$

- Для формулировки вычислительного алгоритма, предположим, что мы уже знаем значение y^{n-1} и y^n . Тогда из (4) мы можем выразить неизвестное значение y^{n+1} :

$$y^{n+1} = 2y^n - y^{n-1} - \tau^2 \omega^2 y^n. \quad (5)$$

Вычисление решения на первом шаге

Очевидно, что для вычисления y^1 необходимо знать неопределенное значение y^{-1} при $t = -\tau$.

Аппроксимируем начальное условие $u'(0) = 0$ центральной разностной производной:

$$\frac{y^1 - y^{-1}}{2\tau} = 0 \Rightarrow y^1 = y^{-1}$$

Подставляя последнее в разностную схему при $n = 0$, получим

$$y^1 = y^0 - \frac{1}{2} \tau^2 \omega^2 y^0. \quad (6)$$

Вычислительный алгоритм

- $y^0 = U$
- вычисляем y^1
- для $n = 1, 2, \dots$,
 - вычисляем y^n

Более строго вычислительный алгоритм напишем на Python:

```
t = linspace(0, T, N+1) # \T2d\cyrs \T2d\cyre \T2d\cyrt \T2d\cyrk \T2
tau = t[1] - t[0]       # \T2d\cyrp \T2d\cyro \T2d\cyrs \T2d\cyrt \T2
u = zeros(N+1)          # \T2d\cyrr \T2d\cyre \T2d\cyrsh \T2d\cyre \T2
u[0] = U
u[1] = u[0] - 0.5*tau**2*omega**2*u[0]
for n in range(1, N):
    u[n+1] = 2*u[n] - u[n-1] - tau**2*omega**2*u[n]
```

Безындексные обозначения

Разностную схему можно записать, используя безындексные обозначения.

Левая и правая разностные производные

$$y_{\bar{t}} \equiv \frac{y^n - y^{n-1}}{\tau}, \quad y_t \equiv \frac{y^{n+1} - y^n}{\tau}.$$

Вторая разностная производная

$$y_{\bar{t}t} \equiv \frac{y_t - y_{\bar{t}}}{\tau} = \frac{y^{n+1} - 2y^n + y^{n-1}}{\tau^2}.$$

Центральная разностная производная:

$$y_t = \frac{y^{n+1} - y^{n-1}}{2\tau}.$$

Функция-решатель (Солвер)

```
def solver(U, omega, tau, T):  
    """  
    \T2A\CYRR \T2A\cyre \T2A\cyrrh \T2A\cyra \T2A\cyre \T2A\cyrt \T2A\  
    u'' + omega**2*u = 0 \T2A\cyrd \T2A\cyrl \T2A\cyrya t \T2A\cyri \  
    \T2A\cyrk \T2A\cyro \T2A\cyrn \T2A\cyre \T2A\cyrch \T2A\cyrn \T2A\  
    """  
    tau = float(tau)  
    Nt = int(round(T/tau))  
    u = np.zeros(Nt+1)  
    t = np.linspace(0, Nt*tau, Nt+1)  
  
    u[0] = U  
    u[1] = u[0] - 0.5*tau**2*omega**2*u[0]  
    for n in range(1, Nt):  
        u[n+1] = 2*u[n] - u[n-1] - tau**2*omega**2*u[n]  
    return u, t
```

Построение графиков

```
def visualize(u, t, U, omega):  
    plt.plot(t, u, 'r--o')  
    t_fine = np.linspace(0, t[-1], 1001) # \T2A\cyrm \T2A\cyre \T2A\  
    u_e = u_exact(t_fine, U, omega)  
    plt.hold('on')  
    plt.plot(t_fine, u_e, 'b-')  
    plt.legend(['\T2A\cyrp \T2A\cyrr \T2A\cyri \T2A\cyrb \T2A\cyrl \T2A\  
    plt.xlabel('$t$')  
    plt.ylabel('$u$')  
    tau = t[1] - t[0]  
    plt.title('$\tau$ \tau = $ %g$ % tau)  
    umin = 1.2*u.min(); umax = -umin  
    plt.axis([t[0], t[-1], umin, umax])  
    plt.savefig('tmp1.png'); plt.savefig('tmp1.pdf')
```

Основная программа

```
U = 1  
omega = 2*pi  
tau = 0.05  
num_periods = 5  
P = 2*np.pi/tau # \T2A\cyro \T2A\cyrd \T2A\cyri \T2A\cyrn \T2A\  
T = P*num_periods  
u, t = solver(U, omega, tau, T)  
visualize(u, t, U, omega, tau)
```

Интерфейс пользователя: командная строка

```
import argparse  
parser = argparse.ArgumentParser()  
parser.add_argument('--U', type=float, default=1.0)  
parser.add_argument('--omega', type=float, default=2*np.pi)  
parser.add_argument('--tau', type=float, default=0.05)  
parser.add_argument('--num_periods', type=int, default=5)  
a = parser.parse_args()  
U, omega, tau, num_periods = a.U, a.omega, a.tau, a.num_periods
```

Стандартный вызов основной программы выглядит следующим образом:

```
Terminal> python vib_undamped.py --num_periods 20 --tau 0.1
```

Простейшие способы тестирования

- Тестирование на простейших решениях: $u = \text{const}$ или $u = ct + d$ не применимы здесь (без правой части в уравнении).
- Ручные вычисления:

вычислить значений y^1, y^2 и y^3 и сравнить с результатом программы.

```
def test_three_steps():  
    from math import pi  
    U = 1; omega = 2*pi; tau = 0.1; T = 1  
    u_by_hand = np.array([  
        1.0000000000000000,  
        0.802607911978213,  
        0.288358920740053])  
    u, t = solver(U, omega, tau, T)  
    diff = np.abs(u_by_hand - u[:3]).max()  
    tol = 1E-14  
    assert diff < tol
```

Оценка скорости сходимости

Чтобы оценить скорость сходимости для рассматриваемой задачи, нужно выполнить

- провести m расчетов, уменьшая на каждом из них шаг в два раза: $\tau_k = 2^{-k}\tau_0$, $k = 0, 1, \dots, m-1$,
- вычислить L_2 -норму погрешности для каждого расчета
$$\varepsilon_k = \sqrt{\sum_{n=0}^{N-1} (y^n - u_e(t_n))^2} \tau_k,$$
- оценить скорость сходимости на основе двух последовательных экспериментов $(\tau_{k-1}, \varepsilon_{k-1})$ и (τ_k, ε_k)

Программная реализация

```
def convergence_rates(m, solver_function, num_periods=8):
    """
    \T2A\CYRV \T2A\cyro \T2A\cyrz \T2A\cyrv \T2A\cyrr \T2A\cyra \T2A\c
    \T2A\cyrp \T2A\cyro \T2A\cyrl \T2A\cyru \T2A\cyrch \T2A\cyre \T2A\c
    \T2A\cyrrh \T2A\cyra \T2A\cyrg \T2A\cyrp \T2A\cyro \T2A\cyru \T2
    solver_function(U, omega, tau, T) \T2A\cyrr \T2A\cyre \T2A\cyrrh \
    \T2A\cyrd \T2A\cyrl \T2A\cyra \T2A\cyrk \T2A\cyro \T2A\cyrt \T2A
    num_periods \T2A\cyrp \T2A\cyre \T2A\cyrr \T2A\cyri \T2A\cyro \T2A
    """
    from math import pi
    omega = 0.35; U = 0.3 # \T2A\cyrp \T2A\cyrr \T2A\cyro \T2A\c
    P = 2*pi/omega # \T2A\cyrp \T2A\cyre \T2A\cyrr \T2A\c
    tau = P/30 # 30 \T2A\cyrrh \T2A\cyra \T2A\cyrg \T
    T = P*num_periods

    tau_values = []
    E_values = []
    for i in range(m):
        u, t = solver_function(U, omega, tau, T)
        u_e = u_exact(t, U, omega)
        E = np.sqrt(tau*np.sum((u_e-u)**2))
        tau_values.append(tau)
        E_values.append(E)
        tau = tau/2

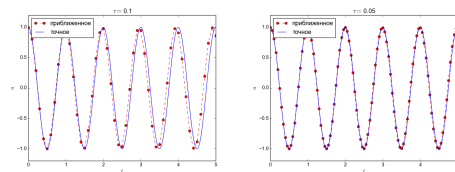
    r = [np.log(E_values[i-1]/E_values[i])/
        np.log(tau_values[i-1]/tau_values[i])]
```

Юнит-тест для скорости сходимости

Используем последнего элемента $r[-1]$ в юнит-тесте:

```
def test_convergence_rates():
    r = convergence_rates(m=5, solver_function=solver, num_periods=8)
    tol = 0.1
    assert abs(r[-1] - 2.0) < tol
```

Влияние шага аппроксимации при моделировании



- Похоже, что численное решение корректно передает амплитуду колебаний
- Наблюдается погрешность при расчете угловой частоты, которая уменьшается при уменьшении шага.
- Суммарная погрешность угловой частоты увеличивается со временем.

Использование изменяющихся графиков

- При моделировании на длительном временном отрезке нам нужно строить график "следующий" за решением.
- Пакет SciTools содержит удобный инструмент для этого: MovingPlotWindow.

Пример вызова:

```
Terminal> python vib_undamped.py --dt 0.05 --num_periods 40
```

Видео

Использование Bokeh для сравнения графиков

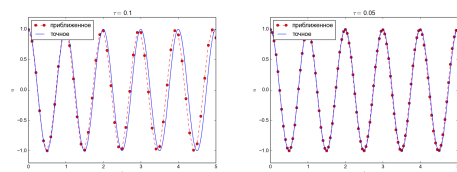
Как выглядит код для построения графиков с помощью Bokeh?

```
def bokeh_plot(u, t, legends, U, omega, t_range, filename):
    """
    \T2A\CYRS \T2A\cyrt \T2A\cyrr \T2A\cyro \T2A\cyri \T2A\cyrt \T2A\cyrs
    \T2A\cyri \T2A\cyrs \T2A\cyrp \T2A\cyro \T2A\cyrl \T2A\cyrsfns \T2A\c
    u \T2A\cyri t - \T2A\cyrs \T2A\cyrp \T2A\cyri \T2A\cyrs \T2A\cyrk \T2
    \T2A\cyrl \T2A\cyre \T2A\cyrg \T2A\cyre \T2A\cyrn \T2A\cyrd \T2A\cyrrer
    """
    if not isinstance(u, (list,tuple)):
        u = [u]
    if not isinstance(t, (list,tuple)):
        t = [t]
    if not isinstance(legends, (list,tuple)):
        legends = [legends]

    import bokeh.plotting as plt
    plt.output_file(filename, mode='cdn', title=u'\T2A\CYRS \T2A\cyrr \T2A\
    # \T2A\CYRP \T2A\cyrr \T2A\cyre \T2A\cyrd \T2A\cyrp \T2A\cyro \T2A\cyr
    t_line = np.linspace(0, t[0][-1], 1001) # \T2A\cyrm \T2A\cyre \T2A\cy
    tools = 'pan,wheel_zoom,box_zoom,reset,\
    'save,box_select,lasso_select'
    u_range = [-1.2*U, 1.2*U]
    font_size = '8pt'
    p = [] # \T2A\cyrs \T2A\cyrp \T2A\cyri \T2A\cyrs \T2A\cyro \T2A\cyrk
    # \T2A\CYRS \T2A\cyro \T2A\cyrs \T2A\cyrd \T2A\cyra \T2A\cyre \T2A\cy
    p_ = plt.figure(
        width=300, plot_height=250, title=legends[0])
```

Анализ конечно-разностной схемы

Можем ли понять появление ошибки в частоте?



Видео с развитием ошибки частоты

Movie 1: [mov-fdm-for-ode/movie.mp4](#)

Можно получить точное решение дискретной задачи

- Мы имеем линейное однородное разностное уравнение относительно y^n
- Решение этого уравнения имеет вид $y^n = Uq^n$, где q — неизвестное число
- Здесь: $u_e(t) = U \cos \omega t \sim U \exp(i\omega t) = U * e^{i\omega \tau n}$
- Прием для упрощения алгебраических выкладок: $y^n = Uq^n$ с $q = \exp(i\tilde{\omega}\tau)$, затем находим $\tilde{\omega}$
- $\tilde{\omega}$ — неизвестная численная частота (проще в вычислении, чем q)
- $\omega - \tilde{\omega}$ — погрешность частоты
- Используем действительную часть как физически обоснованную часть комплексного выражения

Вывод решения конечно-разностной схемы

$$y^n = Uq^n = U \exp(i\tilde{\omega}\tau n) = U \exp(i\tilde{\omega}t) = U \cos(\tilde{\omega}t) + iU \sin(\tilde{\omega}t).$$

$$\begin{aligned} y_{tt}^n &= \frac{y^{n+1} - 2y^n + y^{n-1}}{\tau^2} \\ &= U \frac{q^{n+1} - 2q^n + q^{n-1}}{\tau^2} \\ &= \frac{U}{\tau^2} (e^{i(\tilde{\omega}\tau + \tau)} - 2e^{i\tilde{\omega}\tau} + e^{i(\tilde{\omega}\tau - \tau)}) \\ &= U e^{i\tilde{\omega}\tau} \frac{1}{\tau^2} (e^{i\tau} + e^{-i\tau} - 2) \\ &= U e^{i\tilde{\omega}\tau} \frac{2}{\tau^2} (\cos(\tilde{\omega}\tau) - 1) \\ &= -U e^{i\tilde{\omega}\tau} \frac{4}{\tau^2} \sin^2\left(\frac{\tilde{\omega}\tau}{2}\right) \end{aligned}$$

Решение относительно численной частоты

Подставляя $y^n = Ue^{i\tilde{\omega}\tau n}$ в разностную схему, получим

$$-Ue^{i\tilde{\omega}\tau} \frac{4}{\tau^2} \sin^2\left(\frac{\tilde{\omega}\tau}{2}\right) + \omega^2 Ue^{i\tilde{\omega}\tau} = 0.$$

Разделив последнее выражение на $Ue^{i\tilde{\omega}\tau}$, получим

$$\frac{4}{\tau^2} \sin^2\left(\frac{\tilde{\omega}\tau}{2}\right) = \omega^2.$$

Отсюда

$$\tilde{\omega} = \pm \frac{2}{\tau} \arcsin\left(\frac{\omega\tau}{2}\right). \quad (7)$$

- Численная частота $\tilde{\omega}$ никогда не равна точной ω .
- Насколько хороша аппроксимация?

Полиномиальная аппроксимация погрешности частоты

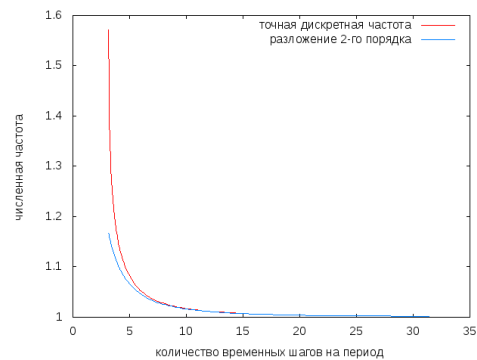
Разложение в ряд Тейлора по τ дает

```
>>> from sympy import *
>>> tau, omega = symbols('tau omega')
>>> omega_tilde_e = 2/tau*asin(omega*tau/2)
>>> omega_tilde_series = omega_tilde_e.series(tau, 0, 4)
>>> print omega_tilde_series
>>> omega + tau**2*omega**3/24 + O(tau**4)
```

$$\tilde{\omega} = \omega \left(1 + \frac{1}{24} \omega^2 \tau^2 \right) + \mathcal{O}(\tau^4). \quad (8)$$

Слишком большая численная частота дает слишком быстро колеблющийся профиль.

График погрешности частоты



Точное дискретное решение

$$y^n = U \cos(\tilde{\omega} \tau n), \quad \tilde{\omega} = \frac{2}{\tau} \arcsin\left(\frac{\omega \tau}{2}\right). \quad (9)$$

Сеточная функция погрешности:

$$\begin{aligned} e^n &= u_e(t_n) - u^n = U \cos(\omega \tau n) - U \cos(\tilde{\omega} \tau n) \\ &= -2U \sin\left(\frac{t}{2}(\omega - \tilde{\omega})\right) \sin\left(\frac{t}{2}(\omega + \tilde{\omega})\right). \end{aligned} \quad (10)$$

Построенная сеточная функция погрешности идеальна для целей тестирования.

Сходимость

Можно легко показать *сходимость*:

$$e^n \rightarrow 0 \text{ при } \tau \rightarrow 0,$$

т.к.

$$\lim_{\tau \rightarrow 0} \tilde{\omega} = \lim_{\tau \rightarrow 0} \frac{2}{\tau} \arcsin\left(\frac{\omega \tau}{2}\right) = \omega.$$

Это можно проверить, например, с помощью *sympy*:

```
>>> import sympy as sym
>>> tau, omega = sym.symbols('tau omega')
>>> sym.limit((2/tau)*sym.asin(omega*tau/2), tau, 0, dir='+')
```

omega

Устойчивость

Наблюдения:

- Численное решение имеет правильную постоянную амплитуду, но ошибку в частоте.
- Постоянная амплитуда будет, если $\arcsin(\omega \tau / 2)$ будет действительным $\Rightarrow |\omega \tau / 2| < 1$
- При $|\omega \tau / 2| > 1$ значения $\arcsin(\omega \tau / 2)$ и, следовательно, $\tilde{\omega}$ являются комплексными:

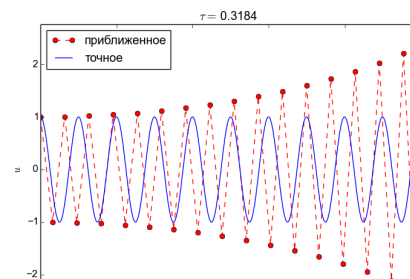
Какие последствия того, что $\tilde{\omega}$ комплексное?

- Пусть $\tilde{\omega} = \tilde{\omega}_r + i\tilde{\omega}_i$.
- Так как $\arcsin(x)$ имеет отрицательную мнимую часть при $x > 1$, $\tilde{\omega}_i < 0$, то $e^{i\tilde{\omega}t} = e^{-\tilde{\omega}_i t} e^{i\tilde{\omega}_r t}$, что означает экспоненциальный рост со временем, так как $e^{-\tilde{\omega}_i t}$ при $\tilde{\omega}_i < 0$ имеет положительную степень.
- Это и есть *неустойчивость*.

Условие устойчивости

$$\frac{\omega \tau}{2} \leq 1 \Rightarrow \tau \leq \frac{2}{\omega}. \quad (11)$$

Возьмем $\tau = \pi^{-1} + 9.01 \cdot 10^{-5}$.



Выводы

Из анализа точности и устойчивости можно сделать некоторые выводы:

- Ключевой параметр в формулах $p = \omega \tau$. Пусть период колебаний $P = 2\pi/\omega$, количество временных шагов на период $N_p = P/\tau$. Тогда $p = \omega \tau = 2\pi N_p$, т.е. основным параметром является количество временных шагов на период. Наименьшее возможное $N_p = 2$, т.е. $p \in (0, \pi]$.
- Если $p \leq 2$, то амплитуда численного решения постоянна.
- Отношение численной угловой частоты к точной есть $\tilde{\omega}/\omega \approx 1 + \frac{p^2}{24}$. Погрешность $\frac{p^2}{24}$ приводит к смещенным пикам численного решения, и эта погрешность расположения пиков растет линейно со временем.

Обобщения: затухание, нелинейные струны и внешние воздействия

$$mu'' + f(u') + s(u) = F(t), \quad t \in (0, T], \quad u(0) = U, \quad u'(0) = V. \quad (12)$$

Здесь m , $f(u')$, $s(u)$, $F(t)$, U , V и T — входные параметры.

Типичные варианты выбора f и s :

- линейное затухание $f(u') = bu'$ или
- квадратичное затухание $f(u') = bu'|u'|$
- линейные пружины: $s(u) = cu$
- нелинейные пружины $s(u) \sim \sin(u)$ (маятник)

Разностная схема для линейного затухания

$$my_{tt}^n + by_t^n + s(y^n) = F^n, \quad n = 1, 2, \dots, N-1, \quad (13)$$

$$y^0 = U, \quad y_t^0 = \frac{y^1 - y^{-1}}{2\tau} = V. \quad (14)$$

В индексной форме:

$$m \frac{y^{n+1} - 2y^n + y^{n-1}}{\tau^2} + b \frac{y^{n+1} - y^{n-1}}{2\tau} + s(y^n) = F^n, \quad n = 1, 2, \dots, N-1, \quad (15)$$

$$y^0 = U, \quad \frac{y^1 - y^{-1}}{2\tau} = V. \quad (16)$$

$$y^{n+1} = \frac{2my^n + (0.5\tau - m)y^{n-1} + \tau^2(F^n - s(y^n))}{m + 0.5b\tau}, \quad n = 1, 2, \dots, N-1. \quad (17)$$

Начальные условия

При $n = 0$ с учетом второго начального условия имеем

$$y^1 = y^0 + \tau V + \frac{\tau^2}{2m}(F^0 - s(y^0) - bV). \quad (18)$$

Разностная схема для квадратичного затухания

- Пусть $f(u') = bu'|u'|$.
- Аппроксимацию $f(u')$ выполним, основываясь на использовании геометрического среднего:

$$(w^2)^n \approx w^{n-1/2} w^{n+1/2},$$

При $w = u'$ имеем

$$(u'|u'|)^n \approx u'(t_{n+1/2})|u'(t_{n-1/2})|.$$

Для аппроксимации u' в точках $t_{n\pm 1/2}$ воспользуемся направленными разностями, которые имеют второй порядок аппроксимации относительно полуцелых точек:

$$u'(t_{n+1/2}) \approx \frac{y^{n+1} - y^n}{\tau} = y_t^n, \quad u'(t_{n-1/2}) \approx \frac{y^n - y^{n-1}}{\tau} = y_t^{n-1}.$$

Разностная схема для квадратичного затухания

Таким образом, получим разностное уравнение

$$my_{tt}^n + by_t^n y^n + s(y^n) = F^n, \quad n = 1, 2, \dots, N-1, \quad (19)$$

которая является линейной относительно y^{n+1} :

$$y^{n+1} = \frac{2my^n - my^{n-1} + by^n|y^n - y^{n-1}| + \tau^2(F^n - s(y^n))}{m + b|y^n - y^{n-1}|}. \quad (20)$$

Начальные условия для квадратичного затухания

Для $t = 0$ имеем $u'(0)|u'(0)| = bV|V|$. Используя это выражение в уравнении и значение $u(0) = U$ при $t = 0$, получим

$$my_{tt}^1 + bV|V| + s(U) = F^0.$$

Отсюда

$$y^1 = y^0 + \tau V + \frac{\tau^2}{2m}(F^0 - mV|V| - s(U)). \quad (21)$$

Алгоритм

- 1 $y^0 = U$;
- 2 вычисляем y^1 (формула зависит от линейного/квадратичного затухания)
- 3 для $n = 1, 2, \dots, N - 1$:
 - 1 вычисляем y^{n+1} (формула зависит от линейного/квадратичного затухания)

Программная реализация

```
def solver(U, V, m, b, s, F, tau, T, damping='linear'):
    """
    |T2A|CYRR |T2A|cyre |T2A|cyrrh |T2A|cyra |T2A|cyre |T2A|cyrt |T2A|
    |T2A|cyrk |T2A|cyro |T2A|cyrn |T2A|cyre |T2A|cyrch |T2A|cyrn |T2A|
    |T2A|CYRE |T2A|cyrs |T2A|cyrl |T2A|cyrl |T2A|cyrz |T2A|cyra |T2A|
    |T2A|cyrt |T2A|cyro |T2A|cyrl |T2A|cyrl |T2A|cyrz |T2A|cyra |T2A|
    |T2A|cyrt |T2A|cyro |T2A|cyrl |T2A|cyrl |T2A|cyrz |T2A|cyra |T2A|
    """
    tau = float(tau); b = float(b); m = float(m) # avoid integer div.
    N = int(round(T/tau))
    u = np.zeros(N+1)
    t = np.linspace(0, N*tau, N+1)

    u[0] = U
    if damping == 'linear':
        u[1] = u[0] + tau*(V + tau**2/(2*m)*(-b*u[0] - s(u[0]) + F(t[0])))
    elif damping == 'quadratic':
        u[1] = u[0] + tau*(V + \
            tau**2/(2*m)*(-b*u[0]*abs(u[0]) - s(u[0]) + F(t[0])))

    for n in range(1, N):
        if damping == 'linear':
            u[n+1] = (2*m*u[n] + (b*tau/2 - m)*u[n-1] + \
                tau**2*(F(t[n]) - s(u[n]))) / (m + b*tau/2)
        elif damping == 'quadratic':
            u[n+1] = (2*m*u[n] - m*u[n-1] + b*u[n]*abs(u[n]) - u[n-1])
```

Верификация реализации алгоритма

- Постоянное решение $u_e(t) = U$ ($V = 0$) удовлетворяет дифференциальной и дискретной задаче. Идеально для отладки!
- Линейная функция $u_e(t) = ct + d$ дифференциальной и дискретной задаче.
- Функция $u_e(t) = bt^2 + Vt + U$ удовлетворяет дифференциальной задаче и дискретной задаче с линейным затуханием, но не с квадратичным. Специальный вид дискретной правой части может допускать u_e как решение дискретной задачи с квадратичным затуханием.

Демонстрационная программа

Сценарий `vib.py` допускает вызов с параметрами командной строки:

```
Terminal> python vib.py --s 'sin(u)' --F '3*cos(4*t)' --b 0.03 --T 60
```

