Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Курсовой проект по курсу

«УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ»

Группа	ПМ-24
Студент(ка)	ПАРАСКУН ИВАН

Новосибирск

2025

Содержание

1. Постановка задачи

Построить МКЭ для уравнения параболического типа

$$-\nabla(\lambda\nabla u) + \gamma u + \sigma\frac{\partial u}{\partial t} = f \tag{1}$$

в декартовой системе координат

$$-\frac{\partial}{\partial x}(\lambda\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(\lambda\frac{\partial u}{\partial y}) - \frac{\partial}{\partial z}(\lambda\frac{\partial u}{\partial z}) + \gamma u + \sigma\frac{\partial u}{\partial t} = f \tag{2}$$

с учётом следующих краевых условий:

$$u\Big|_{S_1} = u_g \tag{3}$$

$$\lambda\frac{\partial u}{\partial n}\Big|_{S_2} = \theta \tag{4}$$

$$\lambda\frac{\partial u}{\partial n}\Big|_{S_3} + \beta(u\Big|_{S_3} - u_\beta) = 0 \tag{5}$$

2. Теоретическая часть

2.1. Вариационная постановка

Эквивалентная вариационная постановка в форме уравнения Галёркина для эллиптического уравнения, входящего в уравнение 1:

$$\begin{aligned}\int_\Omega \lambda\nabla u\nabla v_0 d\Omega + \int_\Omega \gamma u v_0 d\Omega + \int_{S_3} \beta u v_0 dS \\ = \int_\Omega f v_0 d\Omega + \int_{S_2} \theta v_0 dS + \int_{S_3} \beta u_\beta v_0 dS, \quad \forall v_0 \in H_0^1\end{aligned} \tag{6}$$

Аппроксимация уравнения 6 на конечномерных подпространствах V_g^h и V_0^h получается заменой функций $u \in H_g^1$ и $v_0 \in H_0^1$ на функции $u_h \in V_g^h$ и $v_0^h \in V_0^h$ соответственно:

$$\begin{aligned}\int_\Omega \lambda\nabla u^h\nabla v_0^h d\Omega + \int_\Omega \gamma u^h v_0^h d\Omega + \int_{S_3} \beta u^h v_0^h dS \\ = \int_\Omega f v_0^h d\Omega + \int_{S_2} \theta v_0^h dS + \int_{S_3} \beta u_\beta v_0^h dS, \quad \forall v_0^h \in V_0^h\end{aligned} \tag{7}$$

2.2. Конечноэлементная СЛАУ

Раскладывая функции u и v_0 по базису, переходим к конечноэлементной СЛАУ

$$
\sum_{j=1}^{n} \Big(\int_{\Omega} \lambda \nabla \psi_j \nabla \psi_i d\Omega + \int_{\Omega} \gamma \psi_j \psi_i d\Omega + \int_{S_3} \beta \psi_j \psi_i dS \Big) q_j \\
= \int_{\Omega} f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS, \quad i \in N_0 \tag{8}
$$

или в матричном виде

$$
(\mathbf{G} + \mathbf{M}^{\gamma} + \mathbf{M}^{S_3})\mathbf{q} = \mathbf{b} \tag{9}
$$

где компоненты определяются соотношениями

$$
G_{ij} = \int_{\Omega} \lambda \nabla \psi_j \nabla \psi_i d\Omega = \sum_k \int_{\hat{\Omega}} \hat{\lambda} \nabla \hat{\psi}_j \nabla \hat{\psi}_i d\hat{\Omega} \tag{10}
$$

$$
M_{ij}^{\gamma} = \int_{\Omega} \gamma \psi_j \psi_i d\Omega = \sum_k \int_{\hat{\Omega}} \hat{\gamma} \hat{\psi}_j \hat{\psi}_i d\hat{\Omega} \tag{11}
$$

$$
M_{ij}^{S_3} = \int_{S_3} \beta \psi_j \psi_i dS = \sum_k \int_{\hat{S}_3} \hat{\beta} \hat{\psi}_j \hat{\psi}_i d\hat{S} \tag{12}
$$

$$
b_i^{\Omega} = \int_{\Omega} f \psi_i d\Omega = \sum_k \int_{\hat{\Omega}} \hat{f} \hat{\psi}_i d\hat{\Omega} \tag{13}
$$

$$
b_i^{S_2} = \int_{S_2} \theta \psi_i dS = \sum_k \int_{\hat{S}_2} \hat{\theta} \hat{\psi}_i d\hat{S} \tag{14}
$$

$$
b_i^{S_3} = \int_{S_3} \beta u_\beta \psi_i dS = \sum_k \int_{\hat{S}_3} \hat{\beta} \hat{u}_\beta \hat{\psi}_i d\hat{S} \tag{15}
$$

2.3. Трилинейные базисные функции

Трилинейные бизисные функции на прямоугольном параллелепипеде Ω_{psr} строятся следующим образом:

$$
\begin{aligned}
X_1(x) &= \frac{x_{p+1} - x}{h_x}, & X_2(x) &= \frac{x - x_p}{h_x}, & h_x &= x_{p+1} - x_p, \\
Y_1(y) &= \frac{y_{s+1} - y}{h_y}, & Y_2(y) &= \frac{y - y_s}{h_y}, & h_y &= y_{s+1} - y_s, \\
Z_1(z) &= \frac{z_{r+1} - z}{h_z}, & Z_2(z) &= \frac{z - z_r}{h_z}, & h_z &= z_{r+1} - z_r.
\end{aligned}
$$

$$
\hat{\psi}_i = X_{\mu(i)} Y_{\nu(i)} Z_{\vartheta(i)} \tag{16}
$$

Выражения для вычисления компонент локальных матриц и векторов на прямоугольном параллелепипеде Ω_{psr}:

$$\hat{G}_{ij} = \hat{\lambda}(G^x_{\mu(k)\mu(j)\mu(i)} M^y_{\nu(k)\nu(j)\nu(i)} M^z_{\vartheta(k)\vartheta(j)\vartheta(i)} + M^x_{\mu(k)\mu(j)\mu(i)} G^y_{\nu(k)\nu(j)\nu(i)} M^z_{\vartheta(k)\vartheta(j)\vartheta(i)} + M^x_{\mu(k)\mu(j)\mu(i)} M^y_{\nu(k)\nu(j)\nu(i)} G^z_{\vartheta(k)\vartheta(j)\vartheta(i)}) \tag{17}$$

$$\hat{M}^{\gamma}_{ij} = \hat{\gamma}(M^x_{\mu(j)\mu(i)} M^y_{\nu(j)\nu(i)} M^z_{\vartheta(j)\vartheta(i)}) \tag{18}$$

$$\hat{M}^{S_3}_{ij} = \hat{\beta}(M^{\xi}_{\mu(j)\mu(i)} M^{\chi}_{\nu(j)\nu(i)}) \tag{19}$$

$$\hat{b}^{\Omega}_i = \sum_{k=1}^{8} \hat{f}_k (M^x_{\mu(k)\mu(i)} M^y_{\nu(k)\nu(i)} M^z_{\vartheta(k)\vartheta(i)}) \tag{20}$$

$$\hat{b}^{S_2}_i = \sum_{k=1}^{4} \hat{\theta}_k (M^{\xi}_{\mu(k)\mu(i)} M^{\chi}_{\nu(k)\nu(i)}) \tag{21}$$

$$\hat{b}^{S_3}_i = \hat{\beta} \sum_{k=1}^{4} \hat{u}_{\beta,k} (M^{\xi}_{\mu(k)\mu(i)} M^{\chi}_{\nu(k)\nu(i)}) \tag{22}$$

где

$$\mathbf{G}^{\xi} = \frac{1}{h_{\xi}} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{23}$$

$$\mathbf{M}^{\xi} = \frac{h_{\xi}}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{24}$$

2.4. Аппроксимация по времени

2.4.1. Двухслойная схема

Полагая, что коэффициенты λ и γ не зависят от времени, неявная двухслойная схема аппроксимации уравнения 1 по времени может быть выписана как

$$-\nabla(\lambda \nabla u^j) + \gamma u^j + \sigma \frac{u^j - u^{j-1}}{\Delta t} = f^j, \quad j = 1...J \tag{25}$$

где u^{j-1} - решение, полученное на предыдущем временном слое, а u^0 - начальное условие.

Проводя преобразования, аналогичные тем, что были проведены для эллиптической составляющей, получаем следующее матричное уравнение:

$$(\mathbf{G} + \mathbf{M}^{\gamma} + \mathbf{M}^{S_3} + \frac{1}{\Delta t}\mathbf{M}^{\sigma})\mathbf{q}^j = \mathbf{b}^j + \frac{1}{\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-1} \tag{26}$$

2.4.2. Трехслойная схема

Представим искомое решение u на интервале (t_{j-2}, t_j) в следующем виде:

$$u(x,y,z,t) \approx u^{j-2}(x,y,z)\eta_2^j(t) + u^{j-1}(x,y,z)\eta_1^j(t) + u^j(x,y,z)\eta_0^j(t) \tag{27}$$

где функции $\eta_2^j(t), \eta_1^j(t), \eta_0^j(t)$ - базисные квадратичные полиномы Лагранжа, которые могут быть записаны в виде

$$\eta_2^j(t) = \frac{(t - t_{j-1})(t - t_j)}{2\Delta t^2} \tag{28}$$

$$\eta_1^j(t) = -\frac{(t - t_{j-2})(t - t_j)}{\Delta t^2} \tag{29}$$

$$\eta_0^j(t) = \frac{(t - t_{j-2})(t - t_{j-1})}{2\Delta t^2} \tag{30}$$

Производные по времени на j-ом временном слое:

$$\left.\frac{d\eta_2^j(t)}{dt}\right|_{t=t_j} = \frac{1}{2\Delta t} \tag{31}$$

$$\left.\frac{d\eta_1^j(t)}{dt}\right|_{t=t_j} = -\frac{2}{\Delta t} \tag{32}$$

$$\left.\frac{d\eta_0^j(t)}{dt}\right|_{t=t_j} = \frac{3}{2\Delta t} \tag{33}$$

Применяя выражение 27 для аппроксимации производной по времени уравнения 1 на j-ом временном слое:

$$\begin{aligned} &\frac{\partial}{\partial t}(u^{j-2}(x,y,z)\eta_2^j(t) + u^{j-1}(x,y,z)\eta_1^j(t) + u^j(x,y,z)\eta_0^j(t))\Big|_{t=t_j} \\ &\quad - \nabla(\lambda \nabla u^j) + \gamma u^j = f^j, \quad j = 2...J \end{aligned} \tag{34}$$

или, с учётом выражений для вычисления производных

$$-\nabla(\lambda \nabla u^j) + \gamma u^j + \frac{1}{2\Delta t}u^{j-2} - \frac{2}{\Delta t}u^{j-1} + \frac{3}{2\Delta t}u^j = f^j \tag{35}$$

Проводя преобразования, аналогичные тем, что были проведены для эллиптической составляющей, получаем следующее матричное уравнение:

$$(\mathbf{G} + \mathbf{M}^{\gamma} + \mathbf{M}^{S_3} + \frac{3}{2\Delta t}\mathbf{M}^{\sigma})\mathbf{q}^j = \mathbf{b}^j - \frac{1}{2\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-2} + \frac{2}{\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-1} \tag{36}$$

2.4.3. Четырехслойная схема

Представим искомое решение u на интервале (t_{j-2}, t_j) в следующем виде:

$$u(x,y,z,t) \approx u^{j-3}(x,y,z)\eta_3^j(t) + u^{j-2}(x,y,z)\eta_2^j(t) + u^{j-1}(x,y,z)\eta_1^j(t) + u^j(x,y,z)\eta_0^j(t) \quad (37)$$

где функции $\eta_3^j(t), \eta_2^j(t), \eta_1^j(t), \eta_0^j(t)$ - базисные кубические полиномы Лагранжа, которые могут быть записаны в виде

$$\eta_3^j(t) = -\frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{6\Delta t^3} \quad (38)$$

$$\eta_2^j(t) = \frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{2\Delta t^3} \quad (39)$$

$$\eta_1^j(t) = -\frac{(t-t_{j-3})(t-t_{j-2})(t-t_j)}{2\Delta t^3} \quad (40)$$

$$\eta_0^j(t) = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j-1})}{6\Delta t^3} \quad (41)$$

Производные по времени на j-ом временном слое:

$$\left.\frac{d\eta_3^j(t)}{dt}\right|_{t=t_j} = -\frac{1}{3\Delta t} \quad (42)$$

$$\left.\frac{d\eta_2^j(t)}{dt}\right|_{t=t_j} = \frac{3}{2\Delta t} \quad (43)$$

$$\left.\frac{d\eta_1^j(t)}{dt}\right|_{t=t_j} = -\frac{3}{\Delta t} \quad (44)$$

$$\left.\frac{d\eta_0^j(t)}{dt}\right|_{t=t_j} = \frac{11}{6\Delta t} \quad (45)$$

Применяя выражение 37 для аппроксимации производной по времени уравнения 1 на j-ом временном слое:

$$\begin{aligned}&\frac{\partial}{\partial t}(u^{j-3}(x,y,z)\eta_3^j(t) + u^{j-2}(x,y,z)\eta_2^j(t) + u^{j-1}(x,y,z)\eta_1^j(t) + u^j(x,y,z)\eta_0^j(t))\Big|_{t=t_j} \\ &\quad - \nabla(\lambda\nabla u^j) + \gamma u^j = f^j, \quad j = 2...J\end{aligned} \quad (46)$$

или, с учётом выражений для вычисления производных

$$-\nabla(\lambda\nabla u^j) + \gamma u^j - \frac{1}{3\Delta t}u^{j-3} + \frac{3}{2\Delta t}u^{j-2} - \frac{3}{\Delta t}u^{j-1} + \frac{11}{6\Delta t}u^j = f^j \quad (47)$$

Проводя преобразования, аналогичные тем, что были проведены для эллиптической составляющей, получаем следующее матричное уравнение:

$$(\mathbf{G}+\mathbf{M}^{\gamma}+\mathbf{M}^{S_3}+\frac{11}{6\Delta t}\mathbf{M}^{\sigma})\mathbf{q}^j=\mathbf{b}^j+\frac{1}{3\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-3}-\frac{3}{2\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-2}+\frac{3}{\Delta t}\mathbf{M}^{\sigma}\mathbf{q}^{j-1} \tag{48}$$

3. Описание разработанных программ

```
/** Segment. */
typedef struct seg
{
    int vtx[2];
    int pid;
    int qud;
} seg;

/** Quadrangle. */
typedef struct qud
{
    int vtx[4];
    int pid;
    int hxd;
} qud;

/** Hexahedron. */
typedef struct hxd
{
    int vtx[8];
    int pid;
} hxd;

cut_def(seg_cut, seg);
cut_def(qud_cut, qud);
cut_def(hxd_cut, hxd);

/** Unstructured mesh. */
typedef struct msh
{
    struct vec_cut vtx; // vertices
    struct seg_cut seg; // segments
    struct qud_cut qud; // quadrangles
    struct hxd_cut hxd; // hexahedrons
} msh;

int msh_new(struct msh *msh);
int msh_cls(struct msh *msh);

/** Import mesh in Elmer format. */
int msh_imp_grd(struct msh *msh, const char *dir, const char *pfx);

/** Export mesh in CGNS format. */
```

```c
int msh_exp_gns(struct msh *msh, const char *dir, const char *pfx);

/** Calculate quadrangle normal vector. */
int msh_qud_nrm(struct msh *msh, struct qud *qud, struct vec *nrm);

struct sim_fun_ctx
{
    struct sim *sim; // simulation

    int vtx; // hinted vertex
    int qud; // hinted quadrangle
    int hxd; // hinted hexahedron
};

typedef struct val
{
    enum
    {
        VAL_NUM, // constant
        VAL_FUN, // function
    } type;

    union
    {
        double num; // constant value
        mfun   fun; // function of space, time and field
    } as;

    struct
    {
        bool fd;  // field dependence
        mfun dif; // partial derivative with respect to field
    } ops;
} val;

/** Boundary condition. */
typedef struct cnd_bnd
{
    /** Boundary condition type. */
    enum cnd_bnd_type
    {
        CND_BND_DIR, // Dirichlet
        CND_BND_NEU, // Neumann
        CND_BND_ROB, // Robin
    } type;

    union
    {
        struct
        {
            val tgt; // field
        } dir;
```

```
        struct
        {
            val tta; // field flux
        } neu;

        struct
        {
            val bet; // robin coefficient
            val ext; // external field
        } rob;
    } pps;
} cnd_bnd;

/** Initial condition. */
typedef struct cnd_ini
{
    val tgt; // field
} cnd_ini;

cut_def(val_cut, val);
cut_def(cnd_bnd_cut, cnd_bnd);
cut_def(cnd_ini_cut, cnd_ini);

/**
 * Material.
 *
 * Specifies the physical properties of an object.
 */
typedef struct mat
{
    val lam; // diffusion coefficient
    val gam; // reaction coefficient
    val sig;
    val chi;
} mat;

/**
 *  Object.
 *
 *  Physical group for volume elements.
 */
typedef struct obj
{
    int mat; // material
    int ini; // initial condition
    int src; // source field
} obj;

/**
 *  Boundary.
 *
```

```c
 *  Physical group for face elements.
 */
typedef struct bnd
{
    int cnd; // boundary condition
} bnd;

cut_def(mat_cut, mat);
cut_def(obj_cut, obj);
cut_def(bnd_cut, bnd);

/** Simulation. */
typedef struct sim
{
    /** Simulation mode (equation type). */
    enum
    {
        SIM_ELL, // elliptic
        SIM_PBC, // parabolic
        SIM_HYP, // hyperbolic
    } mod;

    struct sim_ops
    {
        /** User-defined functions. */
        struct
        {
            char dir[128]; // usr directory
            char pfx[64];  // usr prefix

            void *hdl; // dynamic-library handler
        } usr;

        struct
        {
            char dir[128]; // mesh directory
            char pfx[64];  // mesh prefix
        } msh;

        /** Export options. */
        struct
        {
            enum
            {
                SIM_EXP_GNS, // CGNS
            } mod;

            char dir[128]; // export directory
            char pfx[64];  // export prefix

            /**
```

```c
             *  Export commons (simulation defined).
             *
             *  @param sim - simulation
             */
            int (*ini)(struct sim *sim);

            /**
             *  Export runtime solution (simulation defined).
             *
             *  @param sim - simulation
             */
            int (*put)(struct sim *sim);
        } exp;

        /** Time discretization options. */
        struct
        {
            int num; // number of time intervals

            double beg; // initial time
            double hop; // time interval length
        } tdd;
    } ops;

    struct msh *msh; // active mesh
    struct slv *slv; // active solver

    struct mat_cut mat; // materials
    struct val_cut src; // sources
    struct obj_cut obj; // objects
    struct bnd_cut bnd; // boundaries

    struct cnd_ini_cut cnd_ini; // initial conditions
    struct cnd_bnd_cut cnd_bnd; // boundary conditions
} sim;

int sim_new(struct sim *sim);
int sim_cls(struct sim *sim);

/** Import simulation from CGNS file. */
int sim_imp_gns(struct sim *sim, const char *gns);

/** Import simulation from Elmer file. */
int sim_imp_elm(struct sim *sim, const char *sif);

/** Export commons in CGNS format. */
int sim_exp_gns_ini(struct sim *sim);

/** Export solution in CGNS format. */
int sim_exp_gns_put(struct sim *sim);
```

```c
/** Start simulation. */
int sim_run(struct sim *sim);

struct apx_fun_ctx
{
    struct sim *sim; // simulation
    struct vec *wgt; // solution (null for runtime)

    int vtx; // hinted vertex
    int qud; // hinted quadrangle
    int hxd; // hinted hexahedron
};

/** Simulation solver. */
typedef struct slv
{
    struct slv_ops
    {
        /** Time discretization strategy. */
        enum tdd_mod
        {
            TDD_I2S = 2, // implicit 2-layered
            TDD_I3S = 3, // implicit 3-layered
            TDD_I4S = 4, // implicit 4-layered
        } tdd;

        /** Options related to initial conditions. */
        struct
        {
            int num; // number of precomputed layers
        } ini;

        /** Options for nonlinear system solver. */
        struct
        {
            enum non_mod
            {
                NON_FPI, // fixed-point iteration
                NON_NEW, // Newton's linearization
            } mod;

            uint8_t map;

            struct non_ops
            {
                int    max; // maximum number of iterations
                double err; // convergence tolerance
                bool   rlx; // enable relaxation

                /**
                 *  Nonlinear iteration callback (user defined).
                 *
```

```c
             *  Called for each nonlinear iteration.
             */
            struct
            {
                void *ctx;
                void (*run)(void *ctx, struct non_ops *ops);
            } itr;

            enum
            {
                DIF_NUM,
                DIF_GIV,
            } dif;

            /** Runtime data made available by solver. */
            struct
            {
                int    itr; // current iteration
                double err; // current error
                double rlx; // optimal relaxation factor
            } run;
        } ops;
    } non;

    /** Options for linear system solver. */
    struct
    {
        enum iss_mod mod;

        union
        {
            struct iss_jac_ops jac;
            struct iss_rlx_ops rlx;
            struct iss_bcg_ops bcg;
        } ops;
    } iss;
} ops;

/**
 *  Execute solver (implementation defined).
 *
 *  @param sim - simulation
 */
int (*exe)(struct sim *sim);

/**
 *  Approximate solution at given point (implementation defined).
 *
 *  @param ctx - context
 *  @param vtx - target point
 */
```

```c
    double (*apx)(struct apx_fun_ctx *ctx, struct vec *vtx);

    /**
     *  Solution callback (user defined).
     *
     *  Called for each time layer. Called once
     *  for elliptic equations.
     *
     *  @param sim - simulation
     */
    struct
    {
        void *ctx;
        void (*run)(void *ctx, struct sim *sim);
    } itr;

    /**
     *  Runtime data made available by solver.
     *
     *  Updated on each time layer and not
     *  available after solver exits.
     */
    struct
    {
        struct vec *wgt[4]; // buffered solution

        int    bs; // number of buffered layers (1 - 4)
        int    ti; // current time iteration
        double tv; // current time value
    } run;
} slv;

/** Finite Element Method simulation solver. */
struct fem
{
    struct slv slv;

    struct fem_ops
    {
        /** Solution mode. */
        enum
        {
            FEM_STD, // standard
            FEM_HMC, // harmonic
        } mod;

        /** Basis functions. */
        enum
        {
            FEM_BSS_LIN, // triliniear
        } bss;
    } ops;
```

```
};

int fem_new(struct fem *fem);
int fem_exe(struct sim *sim);
```

4. Описание тестирования программы

4.1. Тестирование на регурярной сетке

4.1.1. Описание задачи

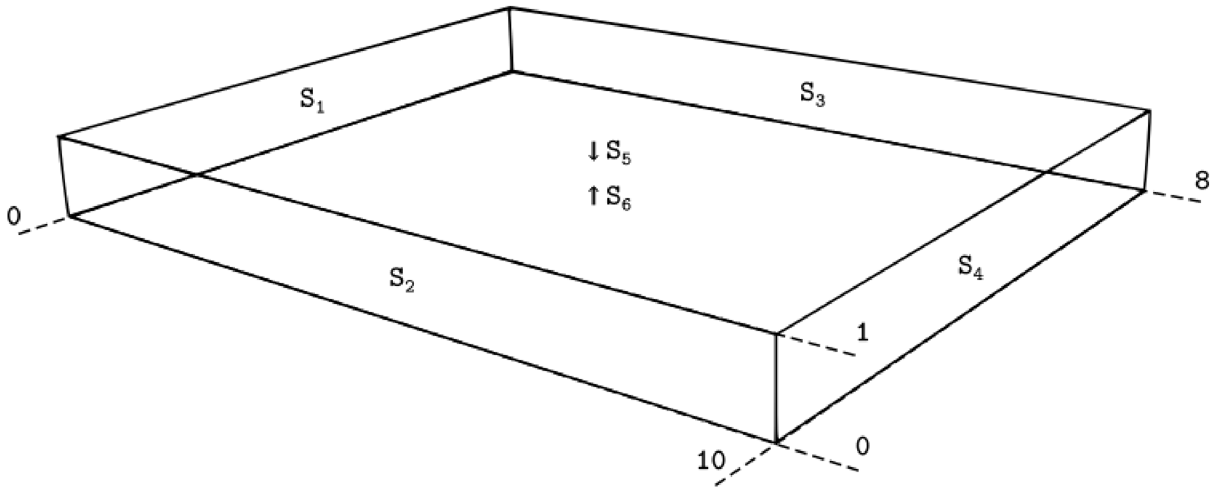

Рис. 1: Засчётная область

Условия

Решение - полином вида

$$u = 5xt^3 - 2zt^2 + yt + 1 \tag{49}$$

Принимая значения коэффициентов $\lambda = 1$, $\gamma = 0$, $\sigma = 2$, $\beta = 10$, получаем

$$\begin{aligned}
&f = 30xt^2 - 8zt + 2y \\
&\theta|_{S_1} = -5t^3 \\
&\theta|_{S_3} = t \\
&\theta|_{S_5} = -2t^2 \\
&u_\beta|_{S_2} = 5xt^3 - 2zt^2 + yt + 1 - 0.1t \\
&u_\beta|_{S_4} = 5xt^3 - 2zt^2 + yt + 1 + 0.5t^3 \\
&u_\beta|_{S_6} = 5xt^3 - 2zt^2 + yt + 1 + 0.2t^2
\end{aligned}$$

Значения параметров по времени: $t_0 = 0$, $t_1 = 25$, $\Delta t = 1$.

4.1.2. Результат выполнения

Для деменстрации работы многослойных схем аппроксимации по времени, решения на первых временных слоях были взяты аналитически (1-3 начальных условия).

j	$E(\|u - u^*\|)$		
	Двухслойная схема	Трёхслойная схема	Четырёхслойная схема
0	0.0000000E+00	0.0000000E+00	0.0000000E+00
1	1.3008431E+02	0.0000000E+00	0.0000000E+00
2	3.9599125E+02	1.0564511E+02	0.0000000E+00
3	7.2839704E+02	1.9439680E+02	3.7414677E-11
4	1.0920167E+03	2.4007804E+02	1.8934309E-11
5	1.4702653E+03	2.5755264E+02	5.3405582E-11
6	1.8553821E+03	2.6224142E+02	7.1549513E-11
7	2.2437265E+03	2.6252408E+02	1.9607504E-10
8	2.6335882E+03	2.6184524E+02	1.2670078E-10
9	3.0241632E+03	2.6127019E+02	3.8652964E-10
10	3.4150735E+03	2.6096067E+02	5.1888799E-10
11	3.8061415E+03	2.6083480E+02	6.3853943E-10
12	4.1972836E+03	2.6079814E+02	1.2761810E-09
13	4.5884605E+03	2.6079426E+02	1.0528181E-09
14	4.9796539E+03	2.6079828E+02	1.3561256E-09
15	5.3708549E+03	2.6080206E+02	1.8517922E-09
16	5.7620596E+03	2.6080417E+02	1.8093863E-09
17	6.1532660E+03	2.6080504E+02	2.1203369E-09
18	6.5444732E+03	2.6080531E+02	2.9247722E-09
19	6.9356808E+03	2.6080535E+02	2.6484560E-09
20	7.3268886E+03	2.6080533E+02	3.3724367E-09
21	7.7180964E+03	2.6080530E+02	3.8412062E-09
22	8.1093043E+03	2.6080529E+02	8.4636231E-09
23	8.5005122E+03	2.6080528E+02	5.3736107E-09
24	8.8917201E+03	2.6080528E+02	1.0244272E-08
25	9.2829280E+03	2.6080528E+02	8.3934569E-09

Таблица 1: Погрешность решения на t_j временном слое

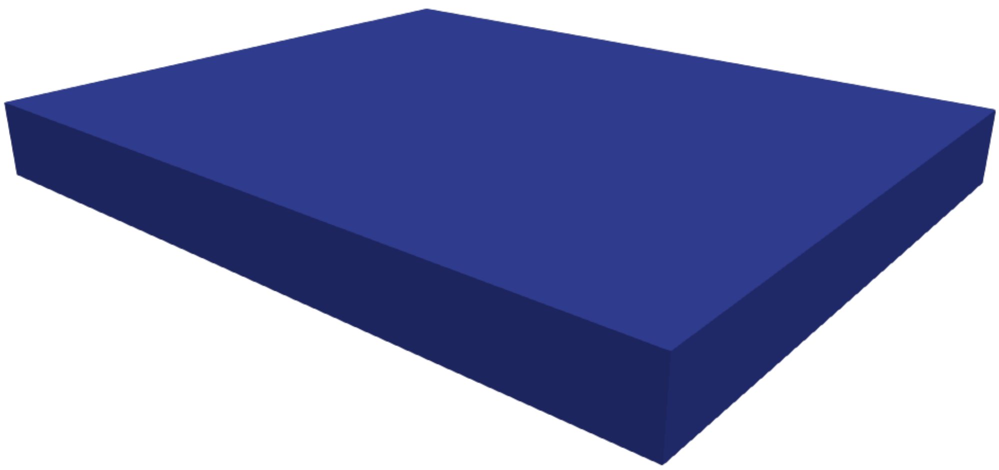

Рис. 2: Визуализация решения

4.2. Тестирование на порядок аппроксимации

Исследование реализованных конфигураций на порядок аппроксимации состоит в отыскании предельной степени решения по времени (k), при котором исследуемый метод даёт точное решение.

4.2.1. Описание задачи

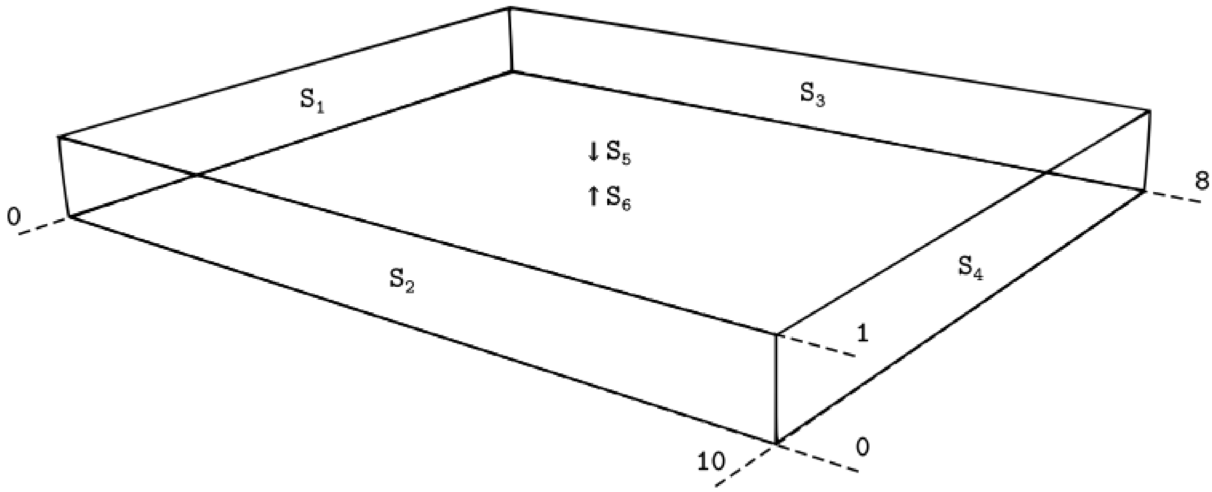

Рис. 3: Засчётная область

Условия

Решение - полином вида

$$u = x\sum_{p=0}^{k}(-1)^p t^p \tag{50}$$

Принимая значения коэффициентов $\lambda = 1$, $\gamma = 0$, $\sigma = 2$, $\beta = 10$, получаем

$$f = \sigma x \sum_{p=1}^{k}(-1)^p p t^{p-1}$$

$$\theta|_{S_1} = -\sum_{p=0}^{k}(-1)^p t^p$$

$$\theta|_{S_3,S_5} = 0$$

$$u_\beta|_{S_2} = x\sum_{p=0}^{k}(-1)^p t^p$$

$$u_\beta|_{S_4} = x\sum_{p=0}^{k}(-1)^p t^p + \frac{1}{\beta}\sum_{p=0}^{k}(-1)^p t^p$$

$$u_\beta|_{S_6} = x\sum_{p=0}^{k}(-1)^p t^p$$

4.2.2. Результат выполнения

k	$E(\|u-u^*\|)$		
	Двухслойная схема	Трёхслойная схема	Четырёхслойная схема
1	1.8910842e-11	1.0456255e-11	1.0443263e-11
2	5.3892025e+00	2.0982637e-11	1.7016870e-11
3	2.1751659e+02	1.3903611e+01	7.1946536e-10
4	3.1009917e+03	9.5079913e+02	4.1711331e+01

Таблица 2: Погрешность решения различных конфигурации в зависимости от степени полинома по времени

4.3. Тестирование на порядок сходимости

Исследование реализованных конфигураций на порядок сходимости состоит в отыскании зависимости между приращением аргумента (Δt) и приращением погрешности ($E(|u-u^*|)$).

4.3.1. Описание задачи

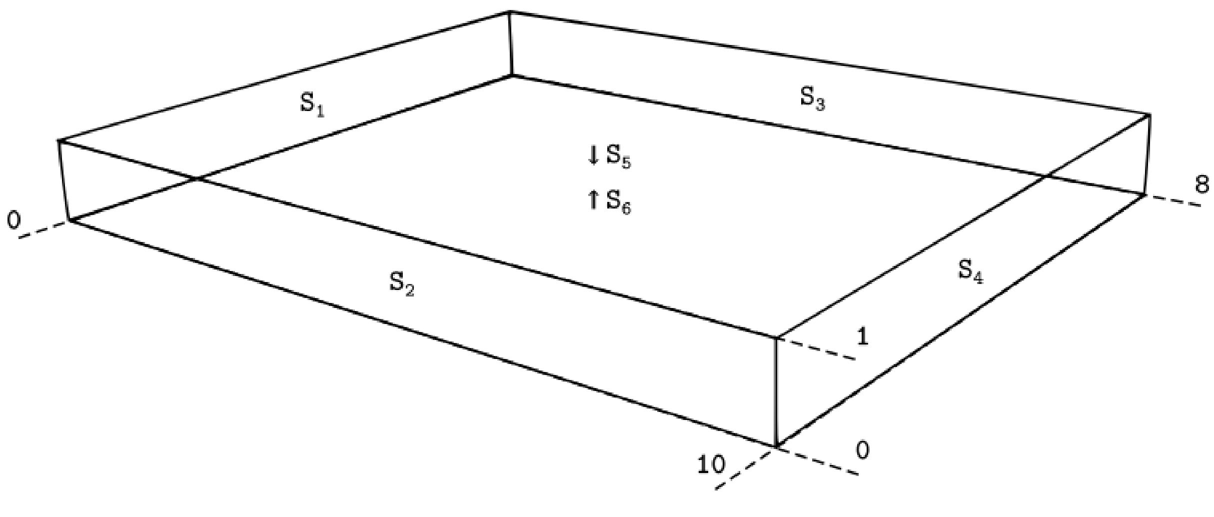

Рис. 4: Засчётная область

Условия

Решение - периодическая функция вида

$$u = xsin(t) \tag{51}$$

Принимая значения коэффициентов $\lambda = 1$, $\gamma = 0$, $\sigma = 2$, $\beta = 10$, получаем

$$f = 2xcos(t)$$
$$\theta|_{S_1} = -sin(t)$$
$$\theta|_{S_3,S_5} = 0$$
$$u_\beta|_{S_2} = xsin(t)$$
$$u_\beta|_{S_4} = xsin(t) + 0.1sin(t)$$
$$u_\beta|_{S_6} = xsin(t)$$

4.3.2. Результат выполнения

Δt	$E(\|u - u^*\|)$		
	Двухслойная схема	Трёхслойная схема	Четырёхслойная схема
128	1.9937299e+00	1.9653237e+00	1.8809388e+00
64	4.0825225e+00	4.1217005e+00	4.1863085e+00
32	2.9847026e+00	2.9443783e+00	2.9496760e+00
16	2.6096649e+00	1.7627706e+00	8.5205136e-01
8	2.9448070e+00	2.2266707e+00	1.2489770e+00
4	6.3049236e+00	6.7250638e+00	7.1964954e+00
2	3.3799714e+00	1.9801666e+00	8.9490448e-01
1	1.7260509e+00	1.4720958e-01	8.8526785e-01
0.5	8.6951389e-01	1.4709618e-01	1.5626740e-01
0.25	4.3448994e-01	6.0461663e-02	1.7622318e-02
0.125	2.2493731e-01	1.8818973e-02	2.0403619e-03
0.0625	1.3608882e-01	3.6531704e-03	2.5589829e-04
0.03125	2.6128831e-02	1.0842462e-03	1.1810640e-05

Таблица 3: Погрешность решения различных конфигурации в зависимости от шага по времени

5. Проведенные исследования и выводы

Порядок аппроксимации двух-, трёх- и четырёхслойной схем равны, соответственно, 1, 2 и 3.

По полученным значениям погрешности решения в зависимости от шага по времени построим зависимость приращения погрешности от шага по времени:

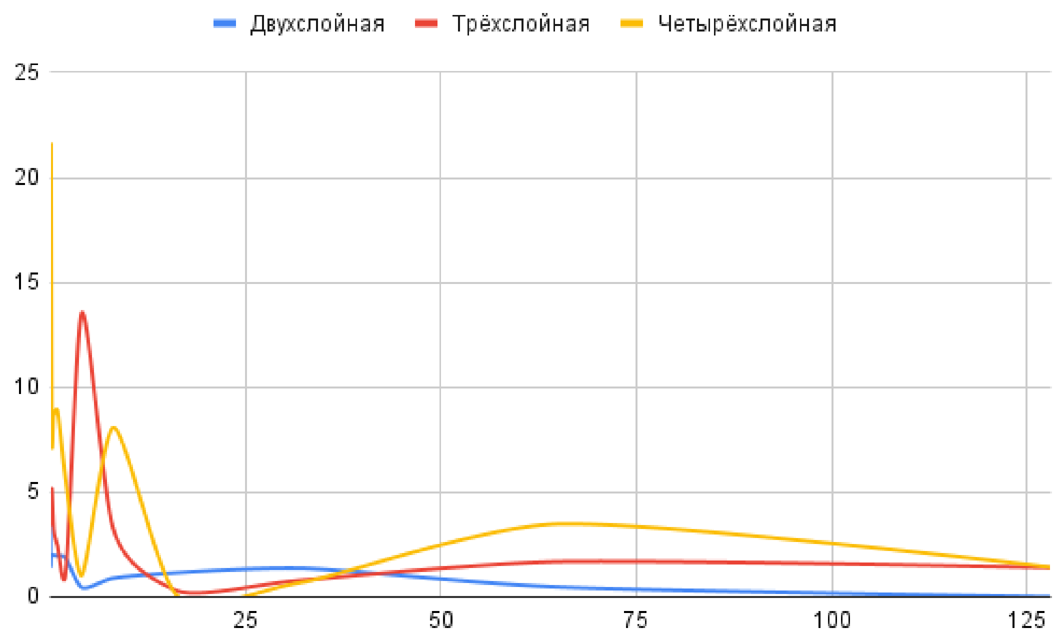

Рис. 5: Зависимость приращения погрешности от шага по времени

6. Приложение

```c
double fem_lin_apx(struct apx_fun_ctx *ctx, struct vec *vtx)
{
    assert(ctx);
    assert(vtx);

    struct vec *v = ctx->sim->msh->vtx.dat;
    struct hxd *h = &ctx->sim->msh->hxd.dat[ctx->hxd];

    double *w = ctx->wgt->dat;

    int v0 = h->vtx[0];
    int v7 = h->vtx[7];

    double x1 = v[v0].dat[0];
    double x2 = v[v7].dat[0];
    double y1 = v[v0].dat[1];
    double y2 = v[v7].dat[1];
    double z1 = v[v0].dat[2];
    double z2 = v[v7].dat[2];

    double hm = (x2 - x1) * (y2 - y1) * (z2 - z1);

    double x = vtx->dat[0];
    double y = vtx->dat[1];
    double z = vtx->dat[2];
    double r = 0;

    r += w[h->vtx[0]] * (x2 - x) * (y2 - y) * (z2 - z) / hm;
    r += w[h->vtx[1]] * (x - x1) * (y2 - y) * (z2 - z) / hm;
    r += w[h->vtx[2]] * (x2 - x) * (y - y1) * (z2 - z) / hm;
    r += w[h->vtx[3]] * (x - x1) * (y - y1) * (z2 - z) / hm;
    r += w[h->vtx[4]] * (x2 - x) * (y2 - y) * (z - z1) / hm;
    r += w[h->vtx[5]] * (x - x1) * (y2 - y) * (z - z1) / hm;
    r += w[h->vtx[6]] * (x2 - x) * (y - y1) * (z - z1) / hm;
    r += w[h->vtx[7]] * (x - x1) * (y - y1) * (z - z1) / hm;

    return r;
}

static int ell_slv(struct sim *sim, struct fem_ctx *ctx);
static int pbc_slv(struct sim *sim, struct fem_ctx *ctx);
static int hyp_slv(struct sim *sim, struct fem_ctx *ctx);

int fem_lin_slv(struct sim *sim, struct fem_ctx *ctx)
{
    assert(sim);

    sim->slv->apx = fem_lin_apx;
```

```c
    switch (sim->mod) {
        case SIM_ELL:
            return ell_slv(sim, ctx);
        case SIM_PBC:
            return pbc_slv(sim, ctx);
        case SIM_HYP:
            return hyp_slv(sim, ctx);
    }

    return 0;
}
static int pbc_i1s_slv(struct sim *sim, struct fem_ctx *ctx);
static int pbc_ctx_shr(struct sim *sim, struct fem_ctx *ctx);

static int pbc_slv(struct sim *sim, struct fem_ctx *ctx)
{
    int r = 0;

    if ((r = vec_new(&ctx->w0, ctx->vec.n)))
        goto end;

    if ((r = vec_new(&ctx->w1, ctx->vec.n)))
        goto end;

    if ((r = vec_new(&ctx->tmp, ctx->vec.n)))
        goto end;

    sim->slv->run.bs = 2;

    if (sim->slv->ops.tdd > 2) {
        if ((r = vec_new(&ctx->w2, ctx->vec.n)))
            goto end;

        sim->slv->run.bs = 3;
    }

    if (sim->slv->ops.tdd > 3) {
        if ((r = vec_new(&ctx->w3, ctx->vec.n)))
            goto end;

        sim->slv->run.bs = 4;
    }

    sim->slv->run.wgt[0] = &ctx->w0;
    sim->slv->run.wgt[1] = &ctx->w1;
    sim->slv->run.wgt[2] = &ctx->w2;
    sim->slv->run.wgt[3] = &ctx->w3;

    double beg = sim->ops.tdd.beg;
    double hop = sim->ops.tdd.hop;
    int    num = sim->ops.tdd.num;
```

```c
    sim->slv->run.tv = beg;

    for (int i = 0; i <= num; ++i) {
        sim->slv->run.ti = i;

        if (i < sim->slv->ops.ini.num) {
            pbc_i1s_slv(sim, ctx);
        } else {
            sys_slv(sim, ctx);
        }

        if (sim->slv->itr.run)
            sim->slv->itr.run(sim->slv->itr.ctx, sim);

        if (sim->ops.exp.put)
            sim->ops.exp.put(sim);

        sim->slv->run.tv += hop;

        pbc_ctx_shr(sim, ctx);
    }

end:
    vec_cls(&ctx->w0);
    vec_cls(&ctx->w1);
    vec_cls(&ctx->w2);
    vec_cls(&ctx->w3);

    vec_cls(&ctx->tmp);

    return r;
}

static int pbc_i1s_slv(struct sim *sim, struct fem_ctx *ctx)
{
    for (int i = 0; i < sim->msh->hxd.len; ++i) {
        struct hxd     *hxd = &sim->msh->hxd.dat[i];
        struct obj     *obj = &sim->obj.dat[hxd->pid];
        struct cnd_ini *ini = &sim->cnd_ini.dat[obj->ini];

        if (ini->tgt.type == VAL_FUN) {
            struct sim_fun_ctx fctx = {
                .sim = sim,
                .vtx = -1,
                .qud = -1,
                .hxd = i,
            };

            for (int k = 0; k < 8; ++k) {
                fctx.vtx = hxd->vtx[k];
```

```
                ctx->w0.dat[hxd->vtx[k]] =
                    ini->tgt.as.fun(&fctx, &sim->msh->vtx.dat[hxd->vtx[k]
            }
        } else {
            for (int k = 0; k < 8; ++k)
                ctx->w0.dat[hxd->vtx[k]] = ini->tgt.as.num;
        }
    }

    return 0;
}

static int pbc_ctx_shr(struct sim *sim, struct fem_ctx *ctx)
{
    struct vec *w0 = &ctx->w0;
    struct vec *w1 = &ctx->w1;
    struct vec *w2 = &ctx->w2;
    struct vec *w3 = &ctx->w3;

    switch (sim->slv->ops.tdd) {
        case TDD_I2S:
            vec_swp(w0, w1);
            break;
        case TDD_I3S:
            vec_swp(w1, w2);
            vec_swp(w0, w1);
            break;
        case TDD_I4S:
            vec_swp(w2, w3);
            vec_swp(w1, w2);
            vec_swp(w0, w1);
            break;
    }

    return 0;
}
int fem_lin_asm(struct sim *sim, struct fem_ctx *ctx)
{
    assert(sim);
    assert(ctx);

    switch (sim->mod) {
        case SIM_ELL:
            return ell_asm(sim, ctx);
        case SIM_PBC:
            return pbc_asm(sim, ctx);
        case SIM_HYP:
            return hyp_asm(sim, ctx);
    }

    return 0;
```

```
}

struct asm_ops
{
    struct smtx *mlam;
    struct smtx *mgam;
    struct smtx *msig;
    struct smtx *mchi;
    struct smtx *mdir;
    struct smtx *mrob;

    struct vec *vsrc;
    struct vec *vdir;
    struct vec *vneu;
    struct vec *vrob;
};

static int assemble(struct sim *sim, struct asm_ops ops);

static int pbc_asm_i2s(struct sim *sim, struct fem_ctx *ctx);
static int pbc_asm_i3s(struct sim *sim, struct fem_ctx *ctx);
static int pbc_asm_i4s(struct sim *sim, struct fem_ctx *ctx);

static int pbc_asm(struct sim *sim, struct fem_ctx *ctx)
{
    int itr = sim->slv->run.ti;

    mtx_rst(&ctx->mtx);
    mtx_rst(&ctx->sig);
    vec_rst(&ctx->vec);

    assemble(sim, (struct asm_ops){
                      .mlam = &ctx->mtx,
                      .mgam = &ctx->mtx,
                      .msig = &ctx->sig,
                      .mchi = NULL,
                      .mdir = NULL,
                      .mrob = NULL,
                      .vsrc = NULL,
                      .vdir = NULL,
                      .vneu = NULL,
                      .vrob = NULL,
                  });

    static int (*f[3])(struct sim *, struct fem_ctx *) = {
        pbc_asm_i2s,
        pbc_asm_i3s,
        pbc_asm_i4s,
    };

    f[min(itr, (int)sim->slv->ops.tdd - 1) - 1](sim, ctx);
```

```c
    return 0;
}

static int pbc_asm_i2s(struct sim *sim, struct fem_ctx *ctx)
{
    // int itr = sim->slv->run.ti;
    double hop = sim->ops.tdd.hop;

    mtx_cmb(&ctx->mtx, &ctx->sig, &ctx->mtx, 1.0 / hop);

    assemble(sim, (struct asm_ops){
                            .mlam = NULL,
                            .mgam = NULL,
                            .msig = NULL,
                            .mchi = NULL,
                            .mdir = &ctx->mtx,
                            .mrob = &ctx->mtx,
                            .vsrc = &ctx->vec,
                            .vdir = &ctx->vec,
                            .vneu = &ctx->vec,
                            .vrob = &ctx->vec,
                      });

    mtx_vmul(&ctx->sig, &ctx->w1, &ctx->tmp);
    vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, 1.0 / hop);

    return 0;
}

static int pbc_asm_i3s(struct sim *sim, struct fem_ctx *ctx)
{
    // int itr = sim->slv->run.ti;
    double hop = sim->ops.tdd.hop;

    mtx_cmb(&ctx->mtx, &ctx->sig, &ctx->mtx, 3.0 / (2 * hop));

    assemble(sim, (struct asm_ops){
                            .mlam = NULL,
                            .mgam = NULL,
                            .msig = NULL,
                            .mchi = NULL,
                            .mdir = &ctx->mtx,
                            .mrob = &ctx->mtx,
                            .vsrc = &ctx->vec,
                            .vdir = &ctx->vec,
                            .vneu = &ctx->vec,
                            .vrob = &ctx->vec,
                      });

    if (mtx_vmul(&ctx->sig, &ctx->w1, &ctx->tmp))
        return -1;
```

```c
    if (vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, 2.0 / hop))
        return -1;

    if (mtx_vmul(&ctx->sig, &ctx->w2, &ctx->tmp))
        return -1;

    if (vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, -1.0 / (2 * hop)))
        return -1;

    return 0;
}

static int pbc_asm_i4s(struct sim *sim, struct fem_ctx *ctx)
{
    // int itr = sim->slv->run.ti;
    double hop = sim->ops.tdd.hop;

    mtx_cmb(&ctx->mtx, &ctx->sig, &ctx->mtx, 11.0 / (6 * hop));

    assemble(sim, (struct asm_ops){
                      .mlam = NULL,
                      .mgam = NULL,
                      .msig = NULL,
                      .mchi = NULL,
                      .mdir = &ctx->mtx,
                      .mrob = &ctx->mtx,
                      .vsrc = &ctx->vec,
                      .vdir = &ctx->vec,
                      .vneu = &ctx->vec,
                      .vrob = &ctx->vec,
                  });

    if (mtx_vmul(&ctx->sig, &ctx->w1, &ctx->tmp))
        return -1;

    if (vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, 3.0 / hop))
        return -1;

    if (mtx_vmul(&ctx->sig, &ctx->w2, &ctx->tmp))
        return -1;

    if (vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, -3.0 / (2 * hop)))
        return -1;

    if (mtx_vmul(&ctx->sig, &ctx->w3, &ctx->tmp))
        return -1;

    if (vec_cmb(&ctx->vec, &ctx->tmp, &ctx->vec, 1.0 / (3 * hop)))
        return -1;
```

```
    return 0;
}
```