СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

УДК 681.3

А.И. Таганов, Д.В. Гильман, В.Г. Псоянц

МЕТОДИКА АТТЕСТАЦИИ УРОВНЕЙ ЗРЕЛОСТИ ПРОЦЕССОВ ПРОГРАММНОГО ПРОЕКТА В УСЛОВИЯХ НЕЧЕТКОСТИ

Предложен формализованный подход к анализу уровней зрелости процессов программного проекта в условиях нечеткости исходных данных, ориентированный на построение алгоритмов и программных средств автоматизации процедур процесса аттестации.

Ключевые слова: *процесс программного проекта, аттестация процесса, уровень зрелости процесса, модель процесса в виде нечеткого графа, алгоритм.*

Введение. На рынке программной продукции наблюдается ранее невиданная конкуренция и практически каждое предприятие-разработчик испытывает большие трудности с получением выгодных заказов, если оно не аттестовано по методологии CMM (Capability Maturity Model) [1]. Заказчики программных проектов требуют от разработчиков обоснованных гарантий технологичности процессов исполнителей и гарантий того, что по способу работы исполнители потенциально не могут оказать некачественную услугу и доказательством этого является аттестация процессов.

На сегодняшний день процесс аттестации по методологии CMM выделен в международный стандарт ИСО 15504 [1], который предлагает общую методику и необходимые модели для аттестации зрелости процессов. Однако применение на практике такой стандартизованной методики в условиях нечеткости исходных данных очень трудоемкая процедура, требующая значительных временных ресурсов, высокой квалификации специалистов-аттестаторов, что затрудняет широкое применение в инженерной практике стандартизованной методики.

Возникает острая необходимость в разработке эффективных программно-методических решений, ориентированных на создание удобных программных сервисов для упрощения и удешевления процедуры применения известной методологии CMM в отечественной практике с несомненным сохранением всех ее достоинств.

Целью работы является разработка подхода к анализу уровней зрелости процессов программного проекта в условиях нечеткости и построение моделей и алгоритмов переработки нечетких данных для формализации процесса аттестации для последующей разработки эффективных программных средств автоматизации.

1. Этапы методики аттестации зрелости процессов проекта. На основе анализа и последующего обобщения научно-методических документов и стандартов международного и национального уровней [1] предлагается к формализации следующая методика аттестации процессов проектной организации или отдельного проекта.

Этап 1 – Определение целевой зрелости процессов проекта.

п. 1. Анализ Исполнителем программного проекта (проектной организацией) заданного технического задания, требований, документов и спецификаций и определение перечня процессов (профиля упорядоченных процессов) программной инженерии, необходимых для реализации заданного проекта. Профиль упорядоченных процессов проекта формируется с учетом правил классификации процессов, представленных в эталонной модели процессов [1].

п. 2. Формирование Заказчиком положения о целевой зрелости (таблица 1), в котором для каждого ключевого процесса постулируется профиль атрибутов процесса и для каждого атрибута постулируется требуемое обладание процесс-

Таблица 1 – Положение о целевой (желаемой) зрелости процессов проекта

Ключевой процесс	*Атрибуты процесса*	*Требуемые рейтинги атрибутов процессов*
Выявление требований	PA1, PA2.1, PA2.2 (т.е. все вплоть до *Управляемого* уровня зрелости включительно)	Обладает полностью
Поддержка потребителя	PA1, PA2.1, PA2.2, PA3.1, PA3.2 (т.е. все вплоть до *Установленного* уровня зрелости включительно)	Обладает полностью
Проектирование программных средств	PA1, PA2.1, PA2.2, PA3.1, PA3.2	Обладает полностью
Конструирование программных средств	PA1, PA2.1, PA2.2, PA3.1, PA3.2	Обладает полностью
	PA4.1, PA4.2	Обладает в основном
Управление проектами	PA1, PA2.1, PA2.2	Обладает полностью
	PA3.1, PA3.2	Обладает в основном
Управление качеством	PA1, PA2.1, PA2.2	Обладает полностью
	PA3.1, PA3.2	Обладает в основном
Управление конфигурацией	PA1.1, PA2.1, PA2.2	Обладает полностью
	PA3.1, PA3.2	Обладает в основном

сом этого атрибута. Целевыми значениями обладания процессом его атрибутов могут быть словесные выражения: *полностью обладает, в основном обладает* или *не требуется* [1, 2].

Этап 2 – Определение уровня зрелости процесса проекта.

п. 3. Выделение очередного процесса из профиля упорядоченных процессов проекта для его последующего анализа и определения уровня зрелости.

п. 4. Выделение очередного атрибута из профиля упорядоченных атрибутов выбранного процесса проекта.

п. 5. Анализ ведущими аттестаторами (экспертами) всей доступной информации по анализируемому процессу и определение экспертным способом *степени обладания атрибута процессом* (САП) проекта. Степень обладания выражается качественно – словами: *полностью обладает, в основном обладает, частично обладает, не обладает*. САП называется рейтингом атрибута процесса, который фиксируется в профиле аттестованной зрелости процессов проекта (рисунок 1). Здесь атрибут процесса представляет собой измеримую характеристику процесса, а рейтинг атрибута процесса – это суждение в рамках определенного контекста процессов о степени, в которой процесс обладает данным атрибутом [1].

п. 6. Если рассмотрены все атрибуты анализируемого процесса, то переход к п. 7, иначе к п. 4.

п. 7. Если рассмотрены все процессы проекта, то переход к этапу 3 (анализ риска), иначе переход к п. 3.

Этап 3 – Анализ риска, сопряженного с процессом. В рамках рассматриваемого подхода к аттестации риска, сопряженного с процессом, вероятность возникновения проблем выводится исходя из того, какой степени разрыв существует между целевой зрелостью и аттестованной зрелостью процесса [1]. На рисунке 2 показан пример сравнения результатов аттестованной зрелости, указанной на рисунке 1 с целевой зрелостью, представленной в качестве примера на рисунке 2.

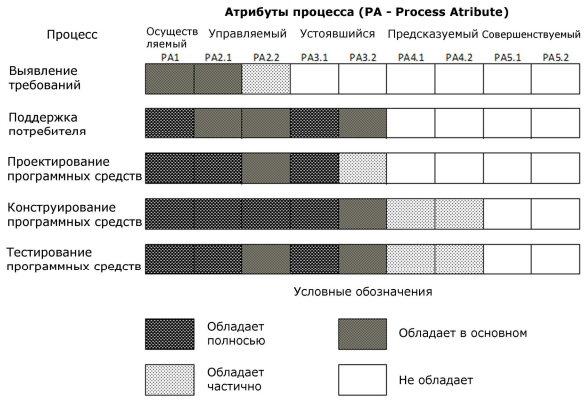


Рисунок 1 – Пример профиля аттестованной зрелости процессов проекта

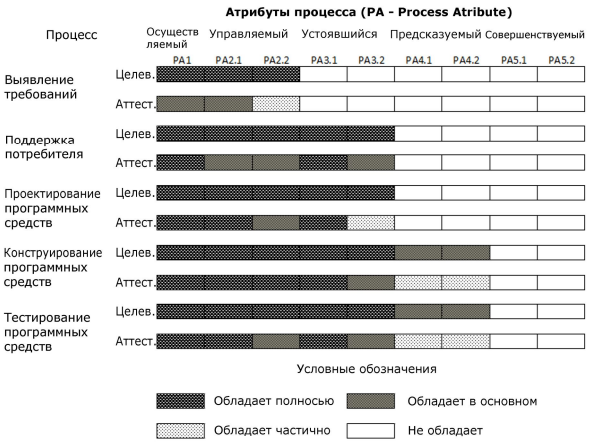


Рисунок 2 – Пример целевой и аттестованной зрелости процессов проекта

Разрыв атрибута процесса возникает, когда конкретный рейтинг атрибута процесса ниже, чем соответствующий рейтинг, заданный в положении о целевой зрелости. Разрывы атрибутов процессов обозначаются в таблице 2.

Таблица 2 – Разрыв атрибутов зрелости

Целевой рейтинг	*Аттестованный рейтинг*	*Разрыв атрибута процесса*
Полностью обладает	Полностью обладает	Нет
	В основном обладает	Небольшой
	Частично обладает	Большой
	Не обладает	Большой
В основном обладает	Полностью обладает	Нет
	В основном обладает	Нет
	Частично обладает	Большой
	Не обладает	Большой

Разрыв уровня зрелости возникает, когда существуют разрывы атрибутов процессов в рамках определенного уровня. Разрывы уровней зрелости обозначаются, как показано в следующей таблице 3.

Таблица 3 – Разрыв уровней зрелости

Количество разрывов атрибутов процессов на уровне зрелости	*Разрыв уровня зрелости*
Нет больших или небольших разрывов	Нет
Только небольшие разрывы	Незначительный
Единственный большой разрыв на уровнях зрелости 2–5	Значительный
Единственный большой разрыв на уровне зрелости 1 или более одного большого разрыва на уровнях зрелости 2–5	Существенный

Как видно из таблицы 3, о возникновении существенного разрыва уровня зрелости говорят при наличии большого разрыва атрибута процесса на уровне зрелости 1, в то время как небольшие разрывы атрибутов процессов на любом из уровней составляют лишь незначительный разрыв уровня зрелости.

Из степени разрыва уровня зрелости процессов проекта экспертами выводятся соответствующие вероятности возникновения проблем в проекте. При этом потенциальное влияние конкретной проблемы зависит от того уровня зрелости, на котором возникает разрыв:

– разрыв на *Совершенствуемом уровне зрелости* может привести к пониженной оптимизации соотношения затрат и времени и к пониженной способности к охвату изменений в технологии;

– разрыв на *Предсказуемом уровне зрелости* может также привести к неспособности предсказывать производительность или своевременно обнаруживать проблемы;

– разрыв на *Устоявшемся уровне зрелости* кроме перечисленных проблем может привести к пониженной эффективности вложений и к пониженной пространственной и временной однородности выполнения;

– разрыв на *Управляемом уровне зрелости* может привести к дальнейшим перерасходам или превышению сроков;

– разрыв на *Выполняемом уровне зрелости* может привести ко всем перечисленным проблемам, а также, что наиболее критично, к отсутствию рабочих продуктов и к неприемлемому качеству продукта.

В рамках рассматриваемого подхода общий сопряженный с процессом риск, связанный с отдельным процессом, может быть обобщен, как показано в таблице 4 [1, 5].

Применяя таблицы 1–3, следует рассмотреть по очереди каждый ключевой процесс, а затем для каждого процесса рассмотреть по очереди каждый уровень зрелости. Следует категоризировать разрывы атрибутов процессов с помощью вербальной информации, представленной в таблице 2, а затем определить разрывы уровней зрелости с помощью таблицы 3. Таблица 4 является лишь примером и руководством по определению общего риска. Фактические уровни риска должны всегда подтверждаться критическими проверками на соответствие опыту и реальности.

Этап 4 – Формирование отчета. Отчет о зрелости процессов – это окончательный результат определения зрелости процессов. Итоговый отчет может состоять из трех частей:

а) введение, в котором описаны контекст определения зрелости процессов, кто проводил это определение и когда, где и почему оно имело место;

б) положение об уверенности заказчика в том, что заявленная зрелость реалистична и, вероятно, на нее можно будет полагаться при удовлетворении заданных требований. Эта уверенность может быть получена из результатов независимой аттестации или из каких-либо иных аспектов взаимоотношений заказчика и организации;

в) отчет, содержащий для каждого ключевого процесса все разрывы между целевой зрелостью и заявленной зрелостью, а также степень сопряженного с процессом риска, возникающего из-за каждого разрыва.

Таблица 5 иллюстрирует, как может быть представлен итоговый отчет о зрелости процессов, показывающий аттестованный общий риск, связанный с каждым процессом.

Таблица 4 – Общий риск, сопряженный с процессом

Положение разрывов уровней зрелости	Степень разрыва уровня зрелости			
	Нет	Незначительный	Значительный	Существенный
Совершенствуемый	Риск не выявлен	Низкая степень риска	Низкая степень риска	Низкая степень риска
Предсказуемый	Риск не выявлен	Низкая степень риска	Низкая степень риска	Средняя степень риска
Устоявшийся	Риск не выявлен	Низкая степень риска	Средняя степень риска	Средняя степень риска
Управляемый	Риск не выявлен	Средняя степень риска	Средняя степень риска	Высокая степень риска
Выполняемый	Риск не выявлен	Средняя степень риска	Высокая степень риска	Высокая степень риска

Таблица 5 – Иллюстрация итогового отчета о зрелости процессов проекта

ИТОГОВЫЙ ОТЧЕТ О ЗРЕЛОСТИ ПРОЦЕССОВ Уверенность в заявленной зрелости		
Степень уверенности в том, что заявленная зрелость реалистична		Достаточная уверенность
Риск, сопряженный с процессом		
Ключевой процесс	Сильные и слабые стороны	Риск, сопряженный с процессом
ENG.1.5	Аттестованная зрелость немного ниже целевой зрелости на Устоявшемся уровне зрелости	Низкая степень риска
CUS.4.2	Аттестованная зрелость немного ниже целевой зрелости на Осуществляемом уровне зрелости, существенно ниже на Управляемом уровне и существенно ниже на Устоявшемся уровне зрелости	Высокая степень риска
SUP.2	Аттестованная зрелость немного ниже целевой зрелости на Управляемом уровне зрелости и значительно ниже на Устоявшемся уровне	Средняя степень риска
ENG.1.4	Аттестованная зрелость соответствует или превышает целевую зрелость во всех аспектах	Риск не выявлен

Итоговый отчет может сопровождаться детальным отчетом, который показывает для каждого процесса целевое и заявленное обладание каждым атрибутом, перечисляет конкретные разрывы атрибутов процессов и получающиеся в итоге разрывы уровней зрелости.

2. Подход к формализации процесса аттестации зрелости процессов проекта. Существенной особенностью рассмотренной методики аттестации процессов является использование на различных этапах аттестации экспертной информации, задаваемой вербальным способом. Это обстоятельство с одной стороны упрощает процесс анализа процессов, но с другой стороны аттестаторы по-разному могут трактовать слова, в смысле тех физических величин, которые они интерпретируют. Эта нечеткость (лингвистическая неопределенность) приводит к неточности в оценках, например, степени обладания атрибутов процессами, а это приводит в итоге к неправильным выводам [2, 4, 6].

Другой особенностью рассмотренной методики аттестации процессов является слишком упрощенная модель процесса на этапе его анализа. Здесь процесс проекта представляется исходя из той описательной информации, которая сопровождает процесс как «черный ящик» и не раскрывает внутренней структуры этого процесса. Это обстоятельство также затрудняет аттестаторам достаточно объективно оценить характеристики процесса.

На этапе анализа рисков, сопряженных с процессом, имеет место принятие аттестаторами решений в условиях отсутствия априорной информации на уровне отдельных процессов и на уровне обобщенного риска проекта тоже. Для формализации этапов аттестации процессов проекта в условиях нечеткости предлагается использовать модели и методы теории нечетких множеств (ТНМ), которые адаптированы к обработке нечеткой информации. Использование из ТНМ такого понятия, как «лингвистическая переменная» [3, 6], позволяет адекватно отразить приблизительное словесное описание важных параметров задачи принятия решений в условиях нечеткости, когда точное описание либо отсутствует, либо является слишком сложным, либо требует больших временных затрат.

Поэтому применение методов ТНМ открывает новое видение принципиального решения важных задач по анализу рисков, сопряженнх с процессами, когда анализируемая информация задана вербальным способом и является нечеткой по определению.

Для решения задачи формализации процесса аттестации модель каждого процесса проекта может быть представлена в виде нечеткого графа, в котором вершинам графа поставлены в соответствие подпроцессы (виды деятельности) процесса, а дугам графа взаимосвязи видов деятельности. Каждой исходящей дуге приписывается вес, определяемый экспертным способом, который характеризует *степень обладания* подпроцессом того атрибута процесса, который входит в перечень атрибутов аттестуемого процесса и оценивается на текущем шаге. Задание весов дуг графа может быть выражено экспертами на основе анализа и оценивания каждого подпроцесса как количественно в интервале [0,1], так и качественно в виде высказываний: *полностью обладает, в основном обладает, частично обладает, не обладает.*

Решение задачи, связанной с определением *степени обладания атрибута процессом,* предлагается найти способом укрупнения (свертывания) нечеткого графа до одной взвешенной дуги. При этом вес этой дуги будет отражать рейтинг атрибута, выражаемый количественно в интервале [0,1] или качественно (*полностью обладает, в основном обладает, частично обладает, не обладает*). Естественно возникают вопросы по применению или разработке необходимых процедур свертывания нечеткого графа при различных способах задания экспертами весов дуг.

Отличительной особенностью предлагаемого подхода к определению уровня зрелости процесса проекта является задание весов дуг графа в виде нечетких чисел в интервале [0,1], по смыслу также отражающих САП из перечня атрибутов эталонной модели процессов. Поэтому для определенности последующих рассуждений и получения необходимых выводов введем определение нечеткого графа процесса проекта.

Нечетким графом (НГ) процесса проекта будем называть конечный ориентированный граф $\tilde{G}(X,\tilde{A})$, дуги которого взвешены нечеткими степенными характеристиками переходов между вершинами. При этом задача укрупнения НГ процесса проекта сводится к преобразованию типа [3, 6] $\tilde{G}(X,\tilde{A}) \to \tilde{G}^*(X,\tilde{A})$, где $\tilde{G}^*(X,\tilde{A})$ – укрупненный НГ, в котором промежуточные вершины отсутствуют, а веса дуг, направленных из вершины x_0 в вершины множества A, представляются в виде функциональных зависимостей от нечетких весов дуг исходного графа.

Наличие укрупненного НГ процесса проекта позволяет вычислить для перехода из x_0 в A *нечеткую степень обладания атрибутов процессом* (НСАП) из заданного перечня атрибутов эталонной модели.

Для укрупнения НГ процесса проекта необходимо разработать правила эквивалентных преобразований для последовательных дуг, параллельных дуг и дуги-петли. При этом характеристику – НСАП (т.е. вес перехода между *i*-й и *j*-й вершинами) будем обозначать через $\tilde{s}_{ij}$.

Для получения правил укрупнения НГ процесса проекта воспользуемся пошаговой методикой обобщения, предложенной и рассмотренной в [3].

Правило «Объединение последовательных дуг НГ». Фрагмент нечеткого графа процесса проекта, состоящий из двух последовательных дуг (рисунок 3), может быть заменен одной эквивалентной дугой, вес которой пересчитывается по формуле:

$$\tilde{s}_{ij} = \bigcup_{\alpha\in[0,1]} (\underline{s}_{ij_\alpha}, \overline{s}_{ij_\alpha}), \qquad (1)$$

где $\overline{s}_{ij_\alpha} = \overline{s}_{ik_\alpha} \cdot \overline{s}_{kj_\alpha}$; $\underline{s}_{ij_\alpha} = \underline{s}_{ik_\alpha} \cdot \underline{s}_{kj_\alpha}$.

Доказательство.

Шаг 1. Исходную модель задаем в виде:

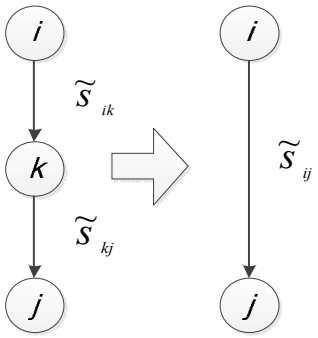


Рисунок 3 – Объединение последовательных дуг

$$s_{ij} = s_{ik} \cdot s_{kj}.$$

Шаг 2. Определяем диапазоны изменения аргумента: $s(\cdot) \in [0,1]$.

Шаг 3. Находим частные производные:

$$\frac{\partial s_{ij}}{\partial s_{ik}} = s_{kj}; \quad \frac{\partial s_{ij}}{\partial s_{kj}} = s_{ik}.$$

Шаг 4. Фиксируем, что все частные производные неотрицательные.

Шаг 5. Получаем правило укрупнения (1).

Правило «Объединение параллельных дуг НГ». Фрагмент НГ процесса проекта, состоящий из двух параллельных дуг (рисунок 4), может быть заменен одной эквивалентной дугой, вес которой пересчитывается по формуле:

$$\tilde{s}_{ij} = \bigcup_{\alpha\in[0,1]} (\underline{s}_{ij_\alpha}, \overline{s}_{ij_\alpha}), \qquad (2)$$

где $\underline{s}_{ij_\alpha} = \frac{\underline{s}'_{ij_\alpha} + \underline{s}''_{ij_\alpha}}{2}$; $\overline{s}_{ij_\alpha} = \frac{\overline{s}'_{ij_\alpha} + \overline{s}''_{ij_\alpha}}{2}$.

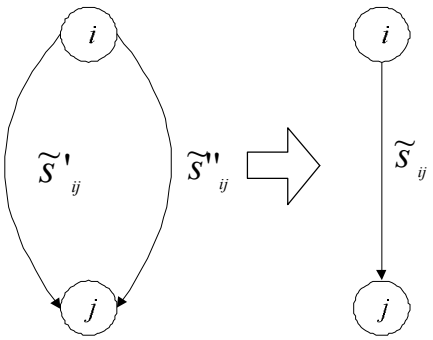


Рисунок 4 – Объединение параллельных дуг

Доказательство.

Шаг 1. Исходную модель задаем в виде:

$$s_{ij} = \frac{s'_{ij} + s''_{ij}}{2}.$$

Шаг 2. Определяем диапазоны изменения аргумента: $s(\cdot) \in [0,1]$.

Шаг 3. Определяем частные производные:

$$\frac{\partial s_{ij}}{\partial s'_{ij}} = \frac{1}{2}; \quad \frac{\partial s_{ij}}{\partial s''_{ij}} = \frac{1}{2}.$$

Шаг 4. Фиксируем, что все частные производные неотрицательные.

Шаг 5. Получаем правило укрупнения (2).

Правило «Удаление дуги-петли НГ». Фрагмент НГ процесса проекта, состоящий из дуги-петли (рисунок 5), может быть заменен одной эквивалентной дугой, вес которой пересчитывается по формуле:

$$\tilde{s}'_{ij} = \bigcup_{\alpha\in[0,1]} (\underline{s}'_{ij_\alpha}, \overline{s}'_{ij_\alpha}), \qquad (3)$$

где $\overline{s}'_{ij_\alpha} = \overline{s}_{ij_\alpha} \cdot \overline{s}_{ii_\alpha}$; $\underline{s}'_{ij_\alpha} = \underline{s}_{ij_\alpha} \cdot \underline{s}_{ii_\alpha}$.

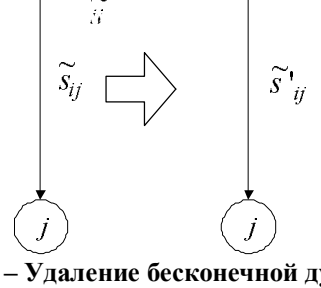


Рисунок 5 – Удаление бесконечной дуги-петли

Доказательство.

Шаг 1. Исходную модель задаем в виде:

$$s'_{ij_\alpha} = s_{ij_\alpha} \cdot s_{ii_\alpha}.$$

Шаг 2. Определяем диапазоны изменения аргумента: $s(\cdot) \in [0,1]$.

Шаг 3. Находим частные производные:

$$\frac{\partial s'_{ij}}{\partial s_{ij}} = s_{ii}; \quad \frac{\partial s'_{ij}}{\partial s_{ii}} = s_{ij}.$$

Шаг 4. Фиксируем, что все частные производные неотрицательные

Шаг 5. Получаем правило укрупнения (3).

Полученные правила используются в дальнейшем для укрупнения нечеткого графа процесса проекта с нечеткими весами дуг.

3. Алгоритмы укрупнения нечеткого графа процесса. Алгоритм укрупнения нечеткого графа процесса проекта, рассматриваемый ниже, базируется на представлении графа в виде так называемой ленточной *L*-матрицы [3, 6].

L-матрицей назовем матрицу размером $3 \times N$, для которой каждая *i*-я строка отождествляется с дугой графа процесса проекта и имеет следующий вид:

$$l_i = \{x_i, y_i, \tilde{s}_i\},$$

где x_i – номер вершины, из которой выходит *i*-я дуга;

y_i – номер вершины, в которую входит *i*-я дуга;

$\tilde{s}_i$ – нечеткая степень обладания атрибутом перехода из вершины x_i в вершину y_i;

N – количество дуг нечеткого графа процесса проекта $(i = \overline{1, N})$.

Идея, лежащая в основе алгоритма укрупнения нечеткого графа процесса, состоит в последовательном применении правил объединения параллельных дуг, удаления дуги-петли и удаления вершины без петли. При этом правила распознавания укрупняемых фрагментов графа с нечеткими весами дуг являются обобщением аналогичных правил для обычного вероятностного графа процесса проекта, рассмотренного выше [3, 6].

Алгоритм правила «Удаление вершины без петли НГ». Обозначим удаляемую вершину через *j*. Тогда дугам, входящим в эту вершину, соответствуют строки *L*-матрицы, второй элемент которых равен *j*. Обозначим эти строки через $a_1, a_2, ..., a_m$, где *m*-количество дуг, входящих в вершину *j*. Дугам, выходящим из вершины *j*, соответствуют строки *L*-матрицы, первый элемент которых равен *j*.

Обозначим эти строки через $b_1, b_2, ..., b_d$, где *d* – количество дуг, выходящих из вершины *j*.

Учитывая формулу (1), получаем следующий алгоритм удаления вершины без петли.

Шаг 1. Выделить множество $A = \{a_1, a_2, ..., a_m\}$ дуг, входящих в вершину *j*, где $a_r = \{x_r, y_r, \tilde{s}_r\}$ – строка *L*-матрицы, $y_r = j$; $r = \overline{1, m}$.

Шаг 2. Выделить множество $B=\{b_1,b_2,...,b_d\}$ дуг, выходящих из вершины j, где $b_z=\{x_z,y_z,\tilde{s}_z\}$ – строка L-матрицы, $x_z=j$; $z=\overline{1,d}$.

Шаг 3. Вычеркнуть из L-матрицы строки, входящие в множество А или В.

Шаг 4. Образовать множество $C=A\times B$, каждый элемент которого представляет собой строку L-матрицы:

$$c_{rz}=\{x_{rz},y_{rz},\tilde{s}_{rz}\},$$

где $x_{rz}=x_r;\ y_{rz}=y_r$.

$\tilde{s}_{rz}=\tilde{s}_r\cdot\tilde{s}_z$ – определяется по формуле (1).

Шаг 5. Добавить в L-матрицу строки, входящие во множество С.

Алгоритм правила «Объединение параллельных дуг НГ». Признаком параллельных дуг является наличие на L-матрице строк, у которых первые два элемента попарно равны.

Учитывая формулу (2), получаем следующий алгоритм для объединения параллельных дуг.

Шаг 1. Найти на L-матрице две строки

$$\{x_i,y_i,\tilde{s}_i\},\ \{x_j,y_j,\tilde{s}_j\}$$

такие, что $x_i=x_j$ и $y_i=y_j$.

Шаг 2. Заменить две строки, найденные на шаге 1, одной эквивалентной строкой $\{x,y,\tilde{s}\}$, где $x=x_i;\ y=y_i$.

$\tilde{s}=\dfrac{\tilde{s}_i+\tilde{s}_j}{2}$ – определяется по формуле (2).

Алгоритм правила «Удаление дуги-петли НГ». Признаком дуги-петли при вершине j есть строка L-матрицы, у которой первый и второй элементы равны j. Дугам, выходящим из вершины j, соответствуют строки, у которых первый элемент равен j.

Учитывая формулу (3), получаем следующий алгоритм удаления дуги-петли при вершине j.

Шаг 1. Определить строку $s=\{x,y,\tilde{s}\}$, для которой $x=y=j$.

Шаг 2. Выделить множество $A=\{a_1,a_2,...,a_m\}$ дуг, выходящих из вершины j, где $a_r=\{x_r,y_r,\tilde{s}_r\}$ – строка L-матрицы, $x_r=j$; $r=\overline{1,m}$, причем $s\notin A$.

Шаг 3. Вычеркнуть из L-матрицы строку s и строки, входящие в множество А.

Шаг 4. Каждую строку a_r, найденную на шаге 2, заменить строкой

$$a_r=\{x'_r,y'_r,\tilde{s}'_r\},$$

где $x'_r=x_r;\ y'_r=y_r;$

$\tilde{s}'_r=\tilde{s}'\cdot\tilde{s}_r$ – определяется по формуле (3);

$r=\overline{1,m}.$.

Шаг 5. Добавить в L-матрицу строки, найденные на шаге 4.

Тогда на основании разработанных правил может быть построен обобщенный алгоритм укрупнения нечеткого вероятностного графа.

Обобщенный алгоритм укрупнения нечеткого графа процесса. Обобщенный алгоритм содержит следующие шаги [3,6].

Шаг 1. Построить граф $\tilde{G}(X,\tilde{A})$ с нечеткими весами дуг.

Шаг 2. Если есть параллельные дуги, то объединить их по правилу «Объединение параллельных дуг НГ».

Шаг 3. Если $\pi=\varnothing$, то взять промежуточную вершину из множества π, иначе перейти к шагу 8.

Шаг 4. Если промежуточная вершина с петлей, то перейти к шагу 5, иначе – к шагу 6.

Шаг 5. Удалить петлю по правилу «Удаление дуги-петли НГ».

Шаг 6. Удалить промежуточную вершину по правилу «Объединение последовательных дуг НГ».

Шаг 7. Перейти к шагу 2.

Шаг 8. Записать результат в виде нечетких весов дуг укрупненного графа $\tilde{G}^*(X,\tilde{A})$.

Таким образом, на основании формализованных правил и представленных алгоритмов появляется возможность разработки программных средств, обеспечивающих эффективную поддержку процесса аттестации уровней зрелости процессов проектной организации или отдельного программного проекта.

Заключение. Предложенный подход к формализации аттестации уровней зрелости проекта на основе графоаналитического способа позволяет автоматизировать трудоемкий этап методики аттестации, связанный с определением степени обладания атрибутов процессом (рейтингов атрибутов) проекта. При этом этап анализа риска, сопряженного с процессом, является достаточно трудоемким и также требует разработки соответствующих формализованных подходов для автоматизации этого этапа. Использование на практике уже автоматизированного процесса оценивания и аттестации процессов проекта или процессов проектной организации способствует выработке культуры постоянного совершенствования процессов и соответствующих механизмов поддержания этой культуры, разработке новых процессов, отвечающих целям наукоемких про-

ектов организации, а также оптимизации использования ресурсов.

Библиографический список

1. ISO/IEC TR 15504 Technical Report. «Information technology – Software process assessment. Part 1–9».

2. *Корячко В.П., Таганов А.И., Таганов Р.А.* Программный метод управления рисками качества проекта информационной системы // Известия Белорусской инженерной академии, 2004. №4. – С. 168–172.

3. *Роттштейн А.П., Штовба С.Д.* Нечеткая надежность алгоритмических процессов. – Винница: Континент-Прим, 1997. – 142 с.

4. *Таганов А.И., Таганов Р.А.* Метод определения важности субъективно связных рисков качества программных проектов // Вестник Рязанского государственного радиотехнического университета. 2002. №10. – С. 59–63.

5. *Таганов А.И., Таганов Р.А.* Методологические основы методов идентификации рисковых событий проекта // Вестник Рязанского государственного радиотехнического университета. 2003. № 12. – С. 70–77.

6. *Таганов А.И., Гильман Д.В.* Задачи и методы нечеткого управления рисками программного проекта // Системы управления и информационные технологии. Москва-Воронеж, 2012. №2(48). – С. 79–83.

УДК 004.272.26

М.А. Козлов, С.В. Скворцов

АЛГОРИТМЫ ПАРАЛЛЕЛЬНОЙ СОРТИРОВКИ ДАННЫХ И ИХ РЕАЛИЗАЦИЯ НА ЯЗЫКЕ CLOJURE

Предложены алгоритмы параллельной сортировки данных, основанные на известных последовательных алгоритмах сортировки слиянием и простыми вставками. Разработаны программные модули для многопоточной реализации алгоритмов сортировки на многоядерных системах средствами языка функционального программирования Clojure. Экспериментально показана целесообразность использования параллельного подхода в практических задачах сортировки.

Ключевые слова: *параллельные алгоритмы, сортировка, многоядерные процессоры.*

Введение. Существует два основных подхода к увеличению производительности процессора. Первый заключается в увеличении тактовой частоты процессора, второй предполагает рост количества инструкций программного кода, выполняемых за один такт процессора.

Увеличение тактовой частоты не может быть бесконечным и определяется технологией изготовления процессора. При этом значение производительности не является прямо пропорциональным тактовой частоте, т.е. наблюдается тенденция насыщаемости, когда дальнейшее увеличение тактовой частоты становится нерентабельным.

Разработка более совершенных архитектур процессоров, содержащих большее число функциональных исполнительных устройств, с целью повышения количества команд, одновременно исполняемых за один такт, – другой традиционный путь повышения производительности, альтернативный росту тактовой частоты. Но такие разработки очень сложны и дороги, причем создание программных модулей для подобных систем требует применения специальных методов планирования параллельных вычислительных процессов [1] и генерации объектного кода [2, 3], а также оптимизирующих компиляторов на их основе.

В настоящее время широкое распространение получили многоядерные процессоры, которые предоставляют большие потенциальные возможности для параллельных вычислений, но также требуют значительных усилий при разработке прикладных программ. Можно сказать, что идея построения многоядерных микропроцессоров является развитием идеи вычислительных кластеров, но в данном случае дублируется целиком процессорное ядро. Другим предшественником многоядерного подхода можно считать технологию Intel HyperThreading, где также есть небольшое дублирование аппаратуры и использование двух потоков инструкций, управляющих общим ядром [4].

Многоядерный процессор имеет два или больше «исполнительных ядер». Операционная система рассматривает каждое из исполнительных ядер, как дискретный процессор со всеми необходимыми вычислительными ресурсами. Поэтому многоядерная архитектура процессора, при поддержке соответствующего программного обеспечения, осуществляет полностью параллельное выполнение нескольких программных потоков. Для реализации процесса параллельного выполнения задач более эффективно интегрировать два ядра или более в одном микропроцессоре. Многоядерная конфигурация, реализованная на одном кристалле, обеспечивает более высокую скорость обмена между ядрами, чем использование внешних шин, коммутаторов и т.п. в многопроцессорных системах.

Таким образом, многоядерная архитектура предоставляет два или более полнофункциональных набора ресурсов для повышения производительности процессора. Многоядерность и 65нм техпроцесс позволили добиться значительной экономии энергопотребления и повышения производительности на 1 Вт потребляемой мощности. Совместно с многоядерными процессорами в архитектуру новых платформ внедряются такие новые технологии, как независимость соответствующих программных компонент (Intel Virtualization Technology VM), ускорение механизма обмена данными (Intel I/O Acceleration Technology), удаленная управляемость (Intel Active Management Technology). Комплекс новых технологий направлен на повышение эффективности вычислительной платформы в целом [4].

Главной тенденцией программирования уже в ближайшее время станет создание новых языков и техник, совмещающих удобство разработки и физический параллелизм исполнения. Например, конкурентное программирование связано с управлением доступом к общим ресурсам из потоков, в свою очередь, параллельное программирование связано с использованием нескольких ядер (процессоров). На сегодняшний день для конкурентного программирования есть хорошо работающие техники и приемы [5]. Кроме этого, существует некоторое количество библиотек к различным языкам программирования, с помощью которых осуществляется управление многопоточностью, например для Java это Concurency, для C – pthread и OpenMP [6]. Однако их использование означает фактически ручное управление физическим параллелизмом, что предполагает привязанность структуры программы к физической архитектуре вычислительной платформы.

Все это требует разработки новых подходов к параллельному программированию, которые направлены на упрощение и повышение эффективности работы программиста.

В данной статье рассматривается возможность использования современных функциональных языков программирования для многопоточной (параллельной) реализации последовательных алгоритмов на примере задачи сортировки данных.

Теоретическая часть. Задача сортировки является одной из наиболее часто встречающихся в практических приложениях. Поэтому ее эффективное решение непосредственно влияет на характеристики программных систем, связанных с обработкой больших объемов данных. Известно достаточно много различных алгоритмов сортировки, отличающихся вычислительной сложностью и затратами памяти. Одним из наиболее быстрых алгоритмов является сортировка слиянием, сложность которого оценивается как $O(n\log_2 n)$ действий, что заметно меньше, чем для методов прямой сортировки массивов, имеющих оценку сложности $O(n^2)$ [7].

Целью данной статьи является разработка алгоритмов параллельного слияния и их многопоточная программная реализация для решения задачи сортировки данных на базе многоядерных процессоров.

Для достижения поставленной цели разработаны алгоритмы сортировки, обеспечивающие параллелизм на уровне данных и использующие возможности функциональных языков программирования для управления многопоточностью. В конечном итоге это позволяет достичь большей производительности при решении задачи сортировки на основе процедуры слияния.

Разработанный *алгоритм параллельного слияния* включает два этапа:

– деление исходного списка на подсписки (этап 1);

– параллельное слияние полученных подсписков (этап 2).

Первый этап не сильно отличается от классического варианта [7], где деление списка на подсписки выполняется рекурсивно. В предлагаемой параллельной реализации первого этапа при рекурсивном спуске каждая итерация слияния выделяется в отдельную нить (pipe) и ожидает выполнения. Как только в подсписках останутся единственные элементы, начинается их слияние, т.е. второй этап алгоритма.

На втором этапе происходит возврат из рекурсии, и каждая пара подсписков сливается в отдельном потоке (thread). Тем самым загружа-

ются все свободные ядра процессора и минимизируется время их простоя.

Представленный алгоритм параллельного слияния демонстрирует идею применения метода «разделяй и властвуй» [4] и является наиболее простым при многопоточной реализации. Для более глубокого исследования характеристик этого алгоритма предлагается еще одна модификация – *гибридный алгоритм параллельного слияния*, который включает дополнительную процедуру сортировки простыми вставками для упорядочения подсписков с малым числом элементов.

Гибридный алгоритм основан на результатах разработки многопоточных алгоритмов сортировки, включая сортировку простыми вставками, которые подробно рассмотрены в работах [8, 9]. Идея этого алгоритма состоит в следующем. Сначала определяется минимальный размер m части списка, которую можно эффективно отсортировать с помощью сортировки вставками в одном потоке. Затем исходный список и все формируемые далее подсписки рекурсивно делятся пополам до тех пор, пока длина полученной части списка превышает величину m. После этого выполняется сортировка каждого подсписка (длина которого не превышает m) с помощью процедуры сортировки простыми вставками. Дальнейшее параллельное слияние упорядоченных подсписков производится так же, как это рассмотрено выше (этап 2).

Заметим, что величина m для каждой конкретной вычислительной системы определяется экспериментально и зависит от таких характеристик, как тактовая частота процессора, количество ядер, размер кэш-памяти и т.п.

Укажем дополнительные достоинства предложенного гибридного алгоритма. Во-первых, он является устойчивым. Во-вторых, сортировка простыми вставками является самой быстрой среди прямых методов для малых размеров сортируемых массивов [7], хотя и имеет оценку сложности $O(n^2)$.

Практическая часть. Для реализации предложенных алгоритмов воспользуемся функциональным языком программирования Clojure [10]. Этот язык представляет собой современный диалект Лиспа и одновременно является языком общего назначения, который поддерживает разработку в интерактивном режиме и упрощает параллельное программирование. В частности, Clojure использует API JVM для работы с многопоточными приложениями, но в отличии от Java он реализует его более эффективно за счет функциональной парадигмы. При этом Clojure обеспечивает неизменность (immutable) данных, что исключает проблемы блокировки ресурсов.

В разработанных программах сортируемые данные представляются в виде связанного однонаправленного списка. Для обработки списка используется функция map, которая представляет собой функцию высшего порядка. Она применяет некоторую функцию к каждому элементу списка и возвращает список результатов. В функциональных языках функция map часто называется «применить ко всем», что достаточно точно определяет ее сущность. В нашем случае, т.е. применительно к задаче сортировки, данная функция будет вызывать функцию слияния ms_merge для каждой части исходного списка.

Деление списка на подсписки осуществляется с помощью функции naive_merge_sort, которая вызывается через функцию map следующим образом:

```
(defn naive_merge_sort [N] (if (< (count N) 2) N
(apply ms_merge
(map naive_merge_sort
(split_at (/ (count N) 2) N)))))
```

Заметим, что деление списка производится рекурсивно с использованием стандартной функции split_at, а переменная *N* определяет длину исходного списка.

Для того чтобы метод "разделяй и властвуй" мог работать в параллельном режиме, будем использовать функцию pmap, что означает «parallel map». Эта функция подобна map, но выполняется параллельно. Она полезна тогда, когда требуется максимально увеличить производительность, несмотря на накладные расходы, связанные с затратами оперативной памяти.

Обращение к функции pmap для деления списка с последующим вызовом функции naive_merge_sort, которая применяется ко всем подспискам, имеет вид:

```
(defn parallel_merge_sort [N] (if (< (count N) 2) N
(apply ms_merge
(pmap naive_merge_sort
(split_at (/ (count L) 2) L)))))
```

Слияние подсписков производится с помощью функции ms_merge, которая получает два подсписка в качестве аргументов:

```
(defn ms_merge [left right] (loop [head [] L left R right]
(if (empty? L) (concat head R)
(if (empty? R) (concat head L) (if (> (first L) (first R))
(recur (conj head (first R)) L (rest R))
(recur (conj head (first L)) (rest L) R))))))
```

В данной функции используется специальная форма рекурсивного вызова (с помощью

стандартной функции recur), а переменные *L* и *R* обозначают «левую» и «правую» части списка.

С использованием перечисленных функций программно реализован алгоритм параллельного слияния. В гибридном алгоритме дополнительно применяется функция сортировки подсписка простыми вставками.

В целом можно указать ряд положительных моментов, которые дает представленная техника программирования:

– единый поток управления программой;

– неявная синхронизация обрабатываемых данных.

Результаты эксперимента. Для оценки полученных результатов проведен эксперимент, целью которого является сравнение скорости работы программ, написанных на языке Clojure и реализующих следующие алгоритмы сортировки:

1) параллельное слияние;
2) гибридный алгоритм;
3) слияние;
4) быстрая сортировка.

Два первых алгоритма предложены в теоретической части данной статьи, а третий и четвертый являются известными последовательными алгоритмами [7], которые используются для сравнительного анализа времени работы разработанных программ.

Исходные данные для сортировки формировались случайным образом и были представлены в виде целочисленных списков достаточно большого размера (10^3, 10^4, 10^5, 10^6 и более элементов). Сравнение соответствующих программ выполнялось по среднему значению времени исполнения, полученному по 10 опытам для каждого размера сортируемого списка. Эксперимент проводился на многоядерных системах, включающих процессоры с 2 и 4 ядрами, функционирующих на одинаковой тактовой частоте под управлением операционной системы Linux.

Результаты эксперимента представлены в таблицах 1 и 2, которые содержат данные о среднем времени выполнения соответствующих программ.

Таблица 1 – Среднее время сортировки (мс) для процессора с двумя ядрами

Алгоритм сортировки	Размер сортируемого списка				
	10^3	10^4	10^5	10^6	3×10^6
Параллельное слияние	367	1259	1446	24513	61404
Гибридный алгоритм	277	842	1627	18253	67539
Слияние	258	1658	6633	18474	72024
Быстрая сортировка	243	1260	2425	27605	70001

Таблица 2 – Среднее время сортировки (мс) для процессора с четырьмя ядрами

Алгоритм сортировки	Размер сортируемого списка				
	10^3	10^4	10^5	10^6	3×10^6
Параллельное слияние	184	1012	1231	19049	58152
Гибридный алгоритм	142	641	1389	14558	62541
Слияние	169	1301	4005	14412	68021
Быстрая сортировка	122	992	1502	21688	64119

Анализ экспериментальных данных для процессора с 4 ядрами позволяет сделать следующие выводы.

1. Сравнение алгоритмов параллельного и последовательного слияния показывает, что для размера списка от 10^3 до 10^4 среднее время многопоточной сортировки резко уменьшается (примерно в 6 раз), но для больших размеров списка ускорение составляет от 20 до 25 %.

2. По сравнению с алгоритмом быстрой сортировки разработанный алгоритм параллельного слияния дает ускорение в 2 – 5 раз для размеров списка от 10^3 до 10^5 элементов и от 10 до 20 % для больших списков (до 3×10^6 элементов).

3. По сравнению с разработанным алгоритмом параллельного слияния гибридный алгоритм дает еще больший прирост производительности (на 20 – 40 %) для списков размером до 10^4 элементов. Кроме того, в целом гибридный алгоритм может конкурировать как с эффективными однопоточными алгоритмами, так и с их параллельными реализациями [8, 9].

Необходимо отметить, что во всех случаях для гибридного алгоритма длина части списка, которая упорядочивается сортировкой вставками, была равна 64 элементам. Эта величина $m = 64$ была подобрана экспериментально с учетом наибольшего выигрыша в скорости сортировки.

Для процессора с 2 ядрами результаты эксперимента аналогичны, но ускорение для больших размеров списка (10^6 и 3×10^6) не так заметно по причине меньшего числа активных ядер.

Заключение. Таким образом, на основе полученных данных в целом можно полагать, что многопоточная реализация алгоритма сортировки слиянием и гибридного алгоритма, дополнительно использующего сортировку простыми вставками, позволяет добиться увеличения быстродействия на больших объемах данных, а на малых объемах дает не худшие результаты по сравнению с однопоточными вариантами известных алгоритмов сортировки. При этом параллельная сортировка слиянием также обладает свойством устойчивости, что важно при решении практических задач.

Библиографический список

1. *Корячко В.П., Скворцов С.В., Телков И.А.* Модель планирования параллельных процессов в суперскалярных процессорах // Информационные технологии. – 1997. – № 1. – С. 8–12.

2. *Скворцов С.В.* Оптимизация кода для суперскалярных процессоров с использованием дизъюнктивных графов // Программирование. 1996. № 2. – С. 41–52.

3. *Скворцов С.В.* Целочисленные модели оптимизации кода по критерию времени // Информационные технологии. 1997. № 10. –С. 2–7.

4. *Робертс Д., Эхтер Ш.* Многоядерное программирование. – СПб.: Питер, 2010. – 320 с.

5. *Богачев К.Ю.* Основы параллельного программирования. – М.: БИНОМ. Лаборатория знаний, 2010. – 342 с.

6. *Камерон Х., Трейси Х.* Параллельное и распределенное программирование с использованием C++. – М.: Вильямс, 2004. – 672 с.

7. *Ахо А.В., Хопкрофт Д., Ульман Д.Д.* Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2010. – 400 с.

8. *Козлов М.А., Скворцов С.В.* Гибридный алгоритм сортировки массивов для многоядерных процессоров // Информационные технологии: Межвуз. сб. / Рязан. гос. радиотехн. ун-т. – Рязань, 2011. – С. 81–84.

9. *Скворцов С.В., Козлов М.А.* Программная реализация алгоритма сортировки для многоядерных процессоров // Информационные технологии в научных исследованиях: Межвуз. сб. / Рязан. гос. радиотехн. ун-т. – Рязань, 2012. – С. 131–133.

10. *Fogus M., Houser C.* The Joy of Clojure: Thinking the Clojure Way. - Manning Publications, 2011. – 300 p.

УДК 537.213

В.С. Иванов, А.А. Трубицын

ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ ФУНКЦИЙ С ИЗЛОМОМ ПРИ ОЦЕНКЕ ГРАДИЕНТА ПОТЕНЦИАЛА В ЭЛЕКТРОННО-ОПТИЧЕСКИХ СИСТЕМАХ

Исследованы стандартные способы численного определения градиента потенциала и предложен модифицированный метод численного дифференцирования, оказавшийся наиболее эффективным в классе решаемых задач. Применение дискретного преобразования Фурье (ДПФ) для вычисления производных от функций с изломом не приводит к ожидаемым положительным результатам. Наибольшую точность показывает метод с использованием левых, центральных и правых разностей, выбор которых осуществляет специальный алгоритм.

Ключевые слова: *градиент потенциала, ДПФ, функция с изломом, МГЭ, ЭОС.*

Введение. Система обыкновенных дифференциальных уравнений движения заряженной частицы в электрическом поле в общем виде выглядит следующим образом

$$\begin{aligned} m\frac{d^2x}{dt^2} &= qE_x + q\left(v_yB_z - v_zB_y\right),\\ m\frac{d^2y}{dt^2} &= qE_y + q\left(v_zB_x - v_xB_z\right),\\ m\frac{d^2z}{dt^2} &= qE_z + q\left(v_xB_y - v_yB_x\right), \end{aligned} \qquad (1)$$

где v_x, v_y, v_z, E_x, E_y, E_z – составляющие вектора скорости частицы, напряженности электрического поля, в общем случае зависящие от времени t; q и m – заряд и масса частицы, $E_x=-\partial U/\partial x$, $E_y=-\partial U/\partial y$, $E_z=-\partial U/\partial z$ - составляющие градиента потенциала.

Опыт показывает, что при наличии эффективных средств вычисления функции распределения потенциала (метод конечных разностей, метод конечных элементов, метод граничных элементов) и проверенных практикой методов траекторного анализа существует узкое место при решении задач компьютерного проектирования электронно-оптических систем в целом – отсутствие приемлемых в отношении обеспечиваемой погрешности методик численной оценки градиентов потенциальных функций. Т.е. задача численного определения градиента потенциала является не до конца разрешенной проблемой.

Цель работы – найти наиболее точный метод численной оценки градиента потенциала в электронно-оптических системах.

Численное определение градиента потенциала. Правая часть дифференциальных уравнений движения (1), представляющая собой силовую функцию воздействия на заряженную частицу, определяется через градиент потенциала. Для плоских и аксиально-симметричных задач потенциал является двумерной функцией $U=U(x,y)$.

Поэтому $\vec{E}(x,y) = gradU = E_x \vec{i} + E_y \vec{j}$, где составляющие градиента $E_x = -\frac{\partial U}{\partial x}$, $E_y = -\frac{\partial U}{\partial y}$. Частные производные в последних выражениях в случае определения функции распределения потенциала численными методами, как в нашем случае методом граничных элементов, могут быть найдены по одномерным формулам численного дифференцирования [1] в узлах сетки с координатами x_j и y_i, где $i=1,2...n_y$; $j=1,2...n_x$. Здесь n_x, n_y – количество узлов сетки разбиения области в направлениях x и y соответственно.

Далее будут исследованы стандартные способы и предложен модифицированный метод численного дифференцирования, оказавшийся наиболее эффективным в классе решаемых задач.

В качестве исходной (эталонной) принята функция распределения потенциала в цилиндрическом конденсаторе:

$$U(r) = V\frac{\ln(r/R1)}{\ln(R2/R1)}, \qquad (2)$$

где V – потенциал внешнего цилиндра, $R1$, $R2$ – радиусы внутреннего и внешнего цилиндров. В нашем случае принималось $V=1$, $R1=1$, $R2=exp(1)$.

При таких параметрах $U(r) = \ln(r)$, а напряженность поля (градиент) между цилиндрами

$$E = -\frac{dU}{dr} = -\frac{1}{r}. \qquad (3)$$

Разрабатываемое компьютерное приложение ориентировано на моделирование систем электронной оптики с реальной толщиной электродов, поэтому на границе электродов функция распределения потенциала будет с изломом, а напряженность поля в этой точке будет испытывать скачок. Для исследований погрешностей численной обработки требуется характерная в указанном классе задач функция. С этой целью в качестве исходной дифференцируемой функции взято распределение потенциала в цилиндрическом конденсаторе (см. рисунок 1, а) при изменении радиальной координаты от 0 до 4. Данная функция $U(r)=0$ для $0 \le r \le R1$, $U(r)=1$ для $R2 \le r \le 4$ и вычисляется по формуле (2) для $R1 \le r \le R2$ (см. рисунок 1, б). График радиальной зависимости модуля градиента потенциала в цилиндрическом конденсаторе в диапазоне $0 \le r \le 4$ с учетом выражения (3) изображен на рисунке 1, в. Последняя функция использована далее в качестве эталонной при сравнении результатов численной оценки производных различными методами.

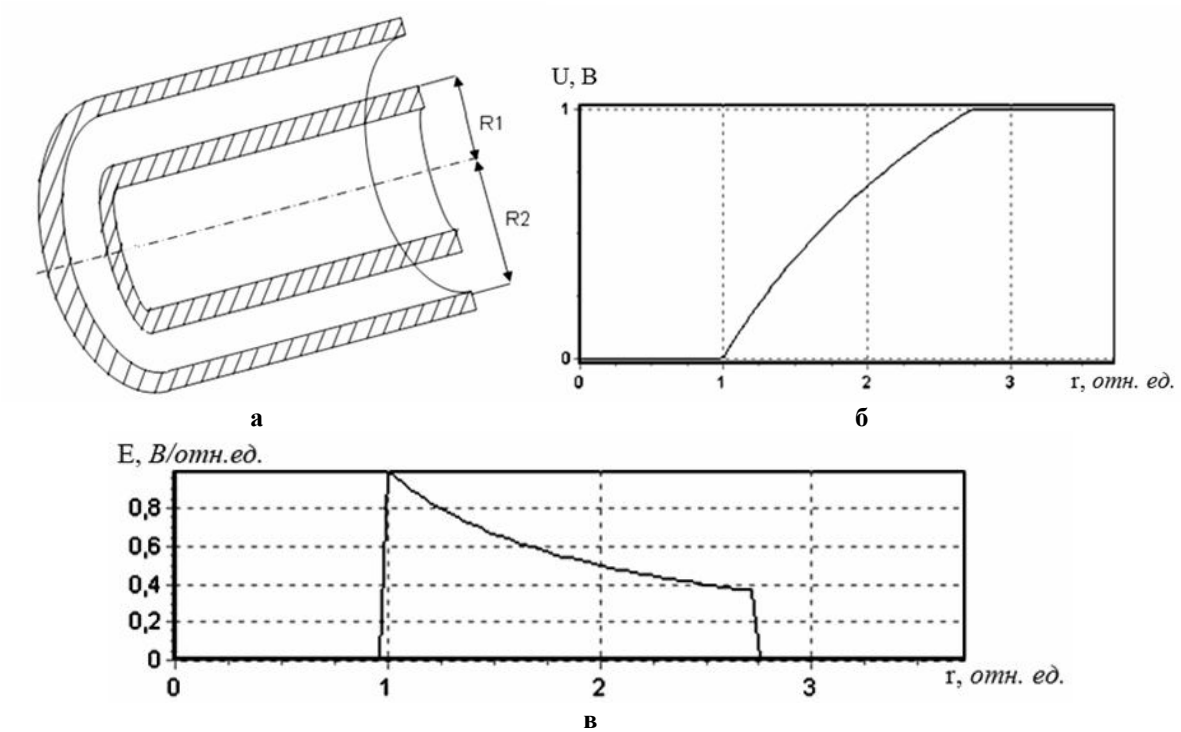


Рисунок 1 – Цилиндрический конденсатор с электродами реальной толщины: а – схематичное изображение конденсатора; б – график радиальной зависимости функции распределения потенциала; в – график модуля градиента потенциала

Исследование точности вычислений градиента с помощью конечно-разностных формул. Метод с использованием интерполяционных формул [2] позволяет получать формулы численного дифференцирования любого порядка точности.

Пусть дана функция $U(r)$ в дискретном наборе n узлов $r_i=i\cdot h$, i=0,1,2 ... (n-1). Записывая, например, интерполяционный многочлен Лагранжа и его остаточный член для случая пяти узлов и вводя обозначение $U(r_i)=U_i$, получаем следующие широко используемые аппроксимации производных [1]:

$$U_0^{'}=\frac{1}{12h}(-25U_0+48U_1-36U_2+16U_3-3U_4)+0(h^4),$$

$$U_1^{'}=\frac{1}{12h}(-3U_0-10U_1+18U_2-6U_3+U_4)-0(h^4),$$

$$U_i^{'}=\frac{1}{12h}(U_{i-2}-8U_{i-1}+8U_{i+1}-U_{i+2})+0(h^4), i=2,3..n-2,$$

$$U_{N-1}^{'}=\frac{1}{12h}(-U_{N-4}+6U_{N-3}-18U_{N-2}+10U_{N-1}+3U_N)-0(h^4),$$

$$U_N^{'}=\frac{1}{12h}(3U_{N-4}-16U_{N-3}+36U_{N-2}-48U_{N-1}+25U_N)+0(h^4),$$

где $N=n-1$.

Использование данных формул позволяет получить результаты, графическое представление которых изображено на рисунке 2. Численные оценки относительных погрешностей применения представленных выше аппроксимаций следующие: средняя погрешность – 0,00739, максимальная погрешность – 0,44255.

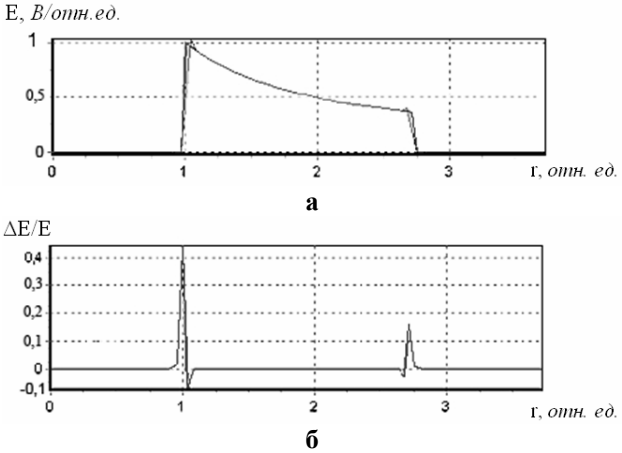


Рисунок 2 – С помощью конечно-разностных формул: а – график численной оценки градиента потенциала; б – погрешность дифференцирования

Как и следовало ожидать, использование стандартных конечно-разностных формул приводит к большим погрешностям вблизи границ электродов *(r=R1 и r=R2)*.

Исследование точности вычислений градиента с помощью модифицированных конечно-разностных формул. Анализ данных численного дифференцирования, полученных по стандартным конечно-разностным формулам, позволяет сделать вывод о низкой точности вычислений. Далее предлагается алгоритм решения поставленной задачи, использующий информацию о поведении дифференцируемой функции на различных участках, где используются конечно-разностные формулы вычисления по левым, правым и центральным разностям.

Формулы вычисления производных по левым разностям

$E_i = U'_i = ELL =$

$= (3U_{i-4} -16U_{i-3}+36U_{i-2} -48U_{i-1}+25U_i)/12h.$

$E_i = U'_i = EL =$

$= (-U_{i-3} +6U_{i-2} -18U_{i-1} +10U_i+3U_{i+1})/12h.$

Формулы вычисления производных по центральным разностям

$E_i=U'_i= EC=(U_{i-2} -8U_{i-1} +8U_{i+1} -U_{i+2})/12h.$

Формулы вычисления производных по правым разностям

$E_i=U'_i=ER=$

$= (-3U_{i-1} -10U_i +18U_{i+1} -6U_{i+2}+U_{i+3})/12h;$

$E_i=U'_i=ERR =$

$= (-25U_i +48U_{i+1} -36U_{i+2}+16U_{i+3}-3U_{i+4})/12h.$

Наиболее точными являются формулы вычислений по центральным разностям.

Если точка расчетной области находится внутри электрода, то ее потенциал равен потенциалу этого электрода, а напряженность поля равна нулю. При расчете электрического поля методом граничных элементов для использования такой (важной) информации создается файл массива точек сетки расчетной области, где каждая точка «помечается» как внутренняя (1) или внешняя (0) по отношению ко всем электродам системы. Далее представляется алгоритм (его суть) определения принадлежности точки какому-либо электроду системы.

Рассмотрим для примера систему, состоящую их двух электродов – электрода 1 и электрода 2, каждый из которых является совокупностью прямолинейных отрезков (рисунок 3).

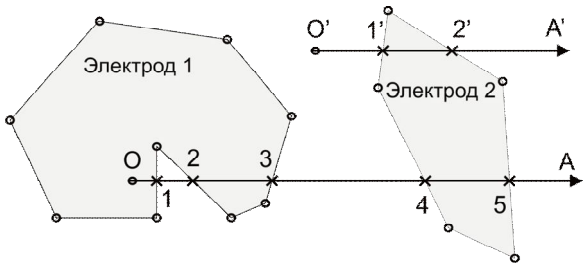


Рисунок 3 – К определению принадлежности точки электроду

Из некоторой точки О проводится вправо луч ОА. Находятся все точки пересечения луча, расположенные правее точки О, с отрезками,

составляющими электроды (границу) системы. В нашем случае это точки 1, 2, 3, 4, 5. Если количество точек пересечения нечетное, то делается вывод о том, что *i*-я точка находится внутри электрода, ей ставится в соответствие «1» (P_i=1) и присваивается потенциал, равный потенциалу электрода с точкой пересечения, ближайшей к точке О. В нашем случае это точка 1 и электрод 1.

Если же количество точек четное, как для луча О'А' на рисунке 3, то это означает, что *i*-я точка (О') находится вне электрода и ей ставится в соответствие «0» (P_i=0).

На основании формул вычисления производных по правым, центральным и левым разностям, а также на основании данных о нахождении точек $r_i=i\cdot h$, i=0,1,2 ... n внутри какого-либо электрода (P_i=0, или P_i=1) разработаны алгоритм и программа вычисления градиента функции:

```
  if P[i]=1 then E[i]:=0
  // i-я точка внутри электрода
(*-------------точка вне электрода-------------*)
  else
  if P[i-2]+P[i-1]+P[i+1]+P[i+2]=0
  then E[i]:=EC // все соседи – вне,
  else
    if P[i-2]+P[i-1]=2
    then E[i]:=ERR // см. рисунок 4, а,
    else
      if P[i-2]=1
      then E[i]:=ER // см. рисунок 4, б,
      else
        if P[i+2]+P[i+1]=2
        then E[i]:=ELL
       // см. рисунок 4, в,
        else
          if P[i+2]=1
          then E[i]:=EL // см. рисунок 4, г.
```

Использование предложенного алгоритма позволяет получить результаты, графическое представление которых изображено на рисунке 4. Численные оценки относительных погрешностей применения представленных выше аппроксимаций следующие (рисунок 5): средняя погрешность – 0,000000744 , максимальная погрешность – 0,000063971.

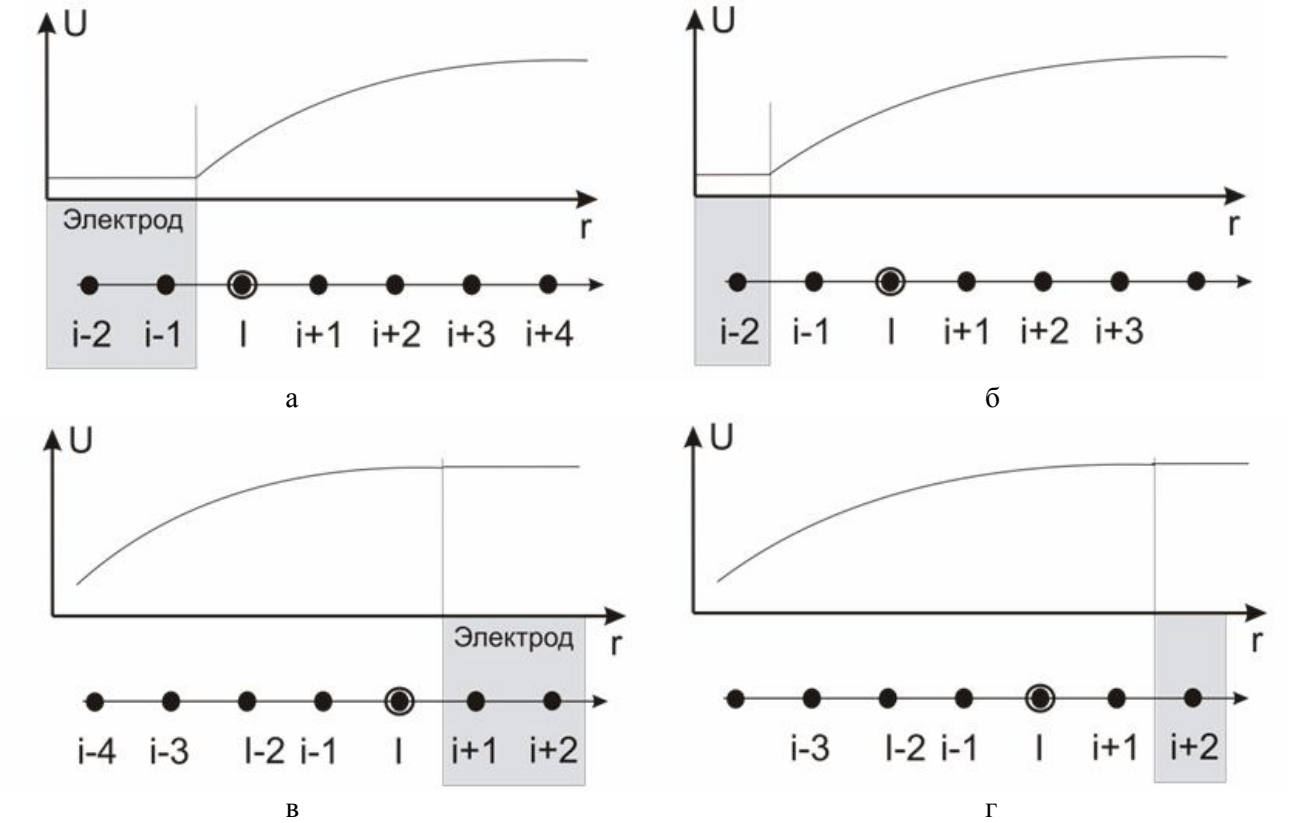
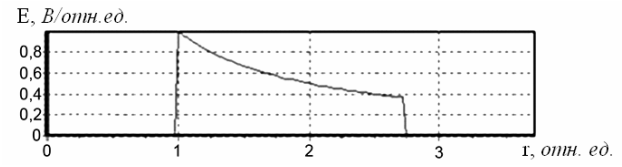
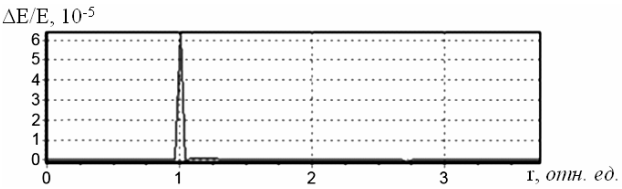


Рисунок 4 – К вычислению градиента потенциала:
а, б – по правым разностям; в, г – по левым разностям



а



б

Рисунок 5 – С помощью модифицированных конечно-разностных формул: а – график численной оценки градиента потенциала; б – погрешность дифференцирования

Исследование точности вычислений градиента с помощью ДПФ. Следует предположить, что достижение лучшей точности численного дифференцирования возможно с помощью использования дискретного преобразования Фурье (ДПФ) [2], поскольку при Фурье-обработке используется вся информация о функции (о значениях функции во всех узлах). Для нахождения производной используем одно из свойств преобразования Фурье, заключающееся в том, что Фурье-образ производной $\langle dU/dr\rangle$ некоторой функции $U(r)$ может быть получен умножением iw на Фурье-образ $\langle U(r)\rangle = \boldsymbol{F}(w)$ этой функции, т.е.

$$\langle dU/dr\rangle = iw\langle U(r)\rangle \rightarrow iw\boldsymbol{F}(w),$$

где w – частота гармонического представления сигнала, i – комплексная единица. Производная функция dU/dr получается обратным Фурье преобразованием Фурье-образа производной $\langle dU/dr\rangle$.

В тригонометрической форме ДПФ Фурье–образ $F_j=F(w_j)$ дискретной функции $U_k=U(r_k)$, $k=0,1\ldots(n-1)$, выражается через амплитуды синусоид (мнимая часть Фурье-образа)

$$B_j = \frac{1}{m}\sum_{k=0}^{n-1} U_k \sin\frac{\pi jk}{m}$$

и амплитуды косинусоид (действительная часть)

$$A_j = \frac{1}{m}\sum_{k=0}^{n-1} U_k \cos\frac{\pi jk}{m},$$

где $F_j = A_j + iB_j$, $m=(n+1)/2$.

В соответствии с выше указанным свойством находим Фурье-образ G_j производной E_k функции U_k ($E=dU/dr$):

$$G_j = iw_j(A_j+iB_j) = -w_jB_j + iw_jA_j = C_j + iD_j,$$

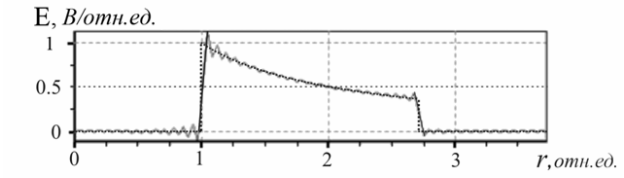
где $C_j=-w_jB_j$ – реальная часть Фурье-образа G, $D_j=w_jA_j$ – мнимая часть G.

Далее по формулам обратного дискретного преобразования Фурье восстанавливаем саму производную E_i

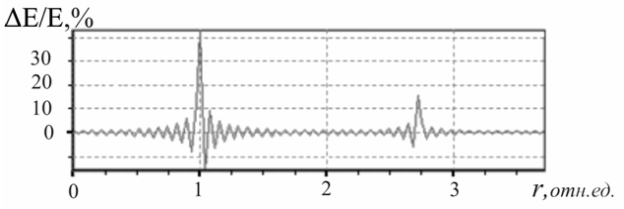
$$E_i = \frac{C_0}{2} + \sum_{j=1}^{m-1}\left[C_j\cos\left(i\frac{\pi j}{m}\right) + D_j\sin\left(i\frac{\pi j}{m}\right)\right] + \frac{C_m}{2}\cos(i\pi),$$

$i=0, 1, \ldots, n-1$.

Результаты применения ДПФ к вычислению производной от тестовой функции U представлены на рисунке 6. Численные оценки относительных погрешностей применения представленных выше аппроксимаций следующие: средняя погрешность оценки производной – 0,025257451, максимальная погрешность – 0,429725423.



а



б

Рисунок 6 – С помощью ДПФ: а – график численной оценки градиента потенциала; б – погрешность дифференцирования

Таким образом, применение ДПФ для вычисления производных от функций с изломом не приводит к ожидаемым положительным результатам. Наибольшую точность показывает предложенный выше метод с использованием левых, центральных и правых разностей, выбор которых осуществляет специальный алгоритм.

Двумерная интерполяция градиента потенциала. Рассмотренные выше методы позволяют определить составляющие (по $0x$ и $0y$) градиента потенциала в узлах двумерной сетки разбиения расчетной области (см. рисунок 7). Однако при интегрировании уравнений движения требуется знание составляющих градиента в произвольных точках области (между узлами). С этой целью необходимо провести двумерную интерполяцию узловых значений составляющих градиента. На прямоугольной сетке, как в нашем случае, удобна *последовательная* интерполяция. Пусть заданы составляющие градиента $E_x(x,y)$ и $E_y(x,y)$ в узлах сетки x_j и y_i. Поскольку в обоих случаях способы реализации двумерной интерполяции полностью идентичны, то в качестве заданной будем рассматривать некоторую функцию $E_{ij}=E(x_j, y_i)$ и находить $E(x,y)$.

Прямоугольная сетка и является расчетной областью ЭОС. Выберем на ней прямоугольник

из $k \cdot k$ узлов (на рисунке 7 величина k=4), в который попадает искомая точка с координатами (x,y), обозначенная на рисунке 7 «кружочком». Для внутренней части расчетной области количества узлов сетки справа и слева, а также сверху и снизу от искомой точки задаются одинаковыми. При приближении к границе области эти количества узлов естественным образом будут отличаться. Отметим, что ближайший к искомой точке узел слева и снизу нумеруется как (i, j).

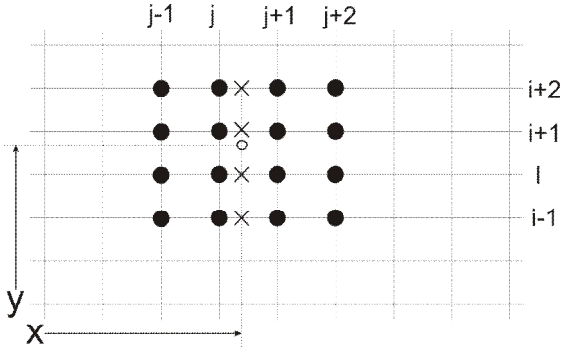


Рисунок 7 – Двумерная интерполяция градиента

Практика показывает, что наилучшими интерполирующими свойствами для функций самого широкого класса обладают сплайны. При последовательной интерполяции сначала проводится одномерная сплайн-интерполяция по строкам, т.е. на каждой из k строк определяются значения функции E в узлах с координатой x (обозначены «крестиками») $E_{i-1}(x)$, $E_i(x)$, $E_{i+1}(x)$, $E_{i+2}(x)$ по k значениям каждой из этих функций на столбцах с номерами j-1, j, j+1, j+2. Затем проводится одномерная сплайн-интерполяция по столбцу (обозначен «крестиками»), т.е. по k значениям $E_{i-1}(x)$, $E_i(x)$, $E_{i+1}(x)$, $E_{i+2}(x)$ находится искомое значение $E(x, y)$.

Ниже представлен текст программы, реализующей алгоритм одномерной сплайн-интерполяции дискретной функции E_i в k=N узлах X_i, i=1, 2 ... N. Оператор-функция *PPVALU* возвращает значение функции в точке с координатой *Xt*. Коэффициенты *Coef* сплайн-представления функции вычисляются с помощью известной и широко используемой процедуры *CUBSPL* [3].

```
Function PPVALU(X,E:TSplineUz; N:integer;
Coef:coefSpl; Xt:double):double;
var k,il,m:integer;
   kmid,h,Gxt:double;
   Procedure INTERV(X:TSplineUz; N:integer;
xt:double; var Left:Integer);
   // определение номера Left ближайшего слева
узла к точке Xt
   var il:integer;
       Xb,Xe:double;
   Begin
      left:=1;
      for il:=1 to N-1 do
      begin
       Xe:=X[il+1];
       Xb:=X[il];
       if Xe>Xb then
       begin
        if (xt>=Xb) and (xt<=Xe) then left:=il;
        if xt>xe then left:=N-1
       end
       else begin
        if (xt>=Xe) and (xt<=Xb) then left:=il;
        if xt<xe then left:=N-1
       end
      end (*il*)
   End; (*int*)

Begin (*ppvalu*)
 k:=4;
 Gxt:=0.;
 kmid:=k;
 INTERV(X,E,N,Xt,il);
 h:=Xt-X[il];
 for m:=k downto 1 do
 begin
  Gxt:=(Gxt/kmid)*h+Coef[m,il];
  kmid:=kmid-1.0
 end; (*m*)
 PPVALU:=Gxt
End; (*ppvalu*)
```

Трехмерная интерполяция градиента аксиально-симметричного потенциала. Из практического опыта следует, что удобнее проводить вычисление траекторий заряженных частиц в трехмерном пространстве в декартовых координатах $0xyz$, что требует знания соответствующих составляющих градиента.

Для планарных полей градиент потенциала по третьей координате z равен нулю.

В случае аксиально-симметричных полей выше описанный алгоритм двумерной интерполяции позволяет восстановить значения составляющих градиента в любой произвольной точке (x,r) меридиональной плоскости $x0r$ – вдоль оси симметрии $0x$ (E_x) и в радиальном направлении $0r$ (E_r). Переход к декартовым координатам не требует серьезных усилий. В каждой точке (y,z) плоскости $y0z$, перпендикулярной к оси симметрии $0x$, y-я и z-я составляющие градиента находятся из выражений (см. рисунок 8)

$$E_y(y,z)=E_r(r)\cdot\cos\varphi=E_r(r)\cdot y/r,$$
$$E_z(y,z)=E_r(r)\cdot\sin\varphi=E_r(r)\cdot z/r,$$

где $r=\sqrt{y^2+z^2}$, а x-я составляющая не зависит от угла φ и является функцией переменных в меридиональной плоскости, т.е. $E_x=E_x(x,r)$.

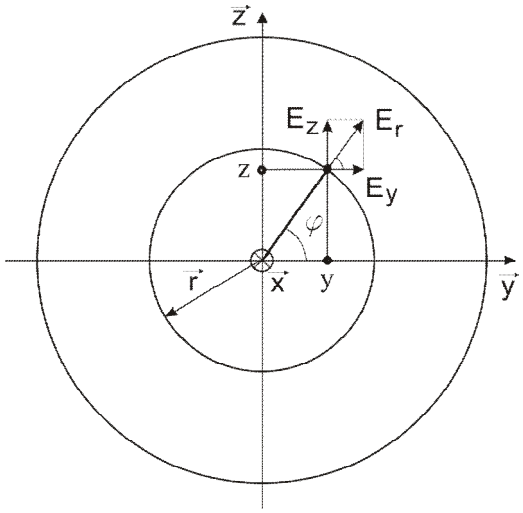


Рисунок 8 – К вычислению составляющих градиента потенциала по декартовым координатам *0xyz*: ось *0x* симметрии ЭОС направлена перпендикулярно к плоскости рисунка «от нас»

Заключение. Исследованы стандартные способы численного определения градиента потенциала и предложен модифицированный метод численного дифференцирования, оказавшийся наиболее эффективным в классе решаемых задач. Применение ДПФ для вычисления производных от функций с изломом не приводит к ожидаемым положительным результатам. Наибольшую точность показывает предложенный в статье метод с использованием левых, центральных и правых разностей, выбор которых осуществляет специальный алгоритм. Для его программной реализации предложены высокоэффективные способы вычислений составляющих градиента электрического потенциала в декартовых координатах, а также его двумерной и трёхмерной интерполяции.

Библиографический список

1. Справочник по специальным функциям под ред. А.Абрамовица, И.Стиган; пер. с англ.; под ред. В.А. Диткина, Л.Н. Карамзиной. – М.: Наука, 1979. – 832 с.

2. *Калиткин Н.Н.* Численные методы. – М.: Наука, 1978. – 512 с.

3. *Карл де Бор*. Практическое руководство по сплайнам. – М.: Радио и связь, 1985. – 304 с.