«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра конструирования и технологии электронно-вычислительных средств

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ В MATHLAB

Методические указания по выполнению лабораторной работы
по дисциплине
«Современные проблемы науки и производства»
направления подготовки магистров 210200.68

Курск 2010

УДК 621.382

Составители: А.В. Кочура

Рецензент
кандидат физико-математических наук, доцент *В.В. Умрихин*

Генетические алгоритмы в MathLab: методические указания по выполнению лабораторной работы по дисциплине «Современные научные проблемы проектирования и технологии электронных средств» / Юго-Зап. гос. ун-т.; сост.: А. В. Кочура, Курск, 2010. 19 с.: ил. 5. табл. 1. Библиогр. с. 19.

Содержатся методические рекомендации по использованию генетических алгоритмов в системе MathLab для решения задач оптимизации.

Указывается порядок выполнения лабораторной работы.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по специальностям автоматики и электроники (УМО АЭ).

Предназначены для студентов направления подготовки магистров 210200.68.

Текст печатается в авторской редакции

Подписано в печать . Формат 60× 84 1/16.
Усл. печ. л. . Уч.-изд. л. . Тираж 30 экз. Заказ . Бесплатно.

Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94

1. ЦЕЛЬ РАБОТЫ

Цель лабораторной работы - закрепление навыков исследования экстремумов функций (для задач оптимизации) с помощью генетических алгоритмов в программной среде MathLab.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Генетические алгоритмы (ГА) относятся к эвристическим алгоритмам поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. ГА является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер. Отличительной особенностью ГА является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Типичными применениями ГА являются следующие: оптимизация функций; оптимизация запросов в базах данных; задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний); настройка и обучение искусственной нейронной сети; задачи компоновки; составление расписаний; игровые стратегии; теория приближений и др.

Основной идеей ГА является организация «борьбы за существование» и «естественный отбор» среди пробных (примерных) решений задачи. Поскольку ГА используют биологические аналогии, то применяющаяся терминология напоминает биологическую.

Как известно, эволюционная теория утверждает, что жизнь на нашей планете возникла вначале лишь в простейших ее формах — в виде одноклеточных организмов. Эти формы постепенно усложнялись, приспосабливаясь к окружающей среде и порождая новые виды, и только через многие миллионы лет появились первые животные и люди. Можно сказать, что каждый биологический вид с течением времени улучшает свои качества так, чтобы наиболее эффективно справляться с важнейшими задачами выживания, самозащиты, размножения и т. д. Таким путем возникла защитная окраска у многих рыб и насекомых, панцирь у черепахи, яд у скорпиона и многие другие полезные приспособления.

С помощью эволюции природа постоянно оптимизирует все живое, находя подчас самые неординарные решения. С первого взгляда неясно, за счет чего происходит этот прогресс, однако ему есть научное объяснение. Дать это объяснение можно, основываясь всего на двух биологических механизмах — естественного отбора и генетического наследования.

Ключевую роль в эволюционной теории играет естественный отбор. Его суть состоит в том, что наиболее приспособленные особи лучше выживают и приносят больше потомства, чем менее приспособленные. Заметим, что сам

по себе естественный отбор еще не обеспечивает развития биологического вида. Действительно, если предположить, что все потомки рождаются примерно одинаковыми, то различные поколения будут отличаться только по численности, но не по приспособленности. Поэтому очень важно изучить, каким образом происходит наследование, т. е. как свойства потомка зависят от свойств родителей.

Основной закон наследования интуитивно понятен каждому - он состоит в том, что потомки похожи на родителей. В частности, потомки более приспособленных родителей будут, скорее всего, одними из наиболее приспособленных в своем поколении. Чтобы понять, на чем основана эта похожесть, нам потребуется немного углубиться в строение животной клетки - в мир генов и хромосом.

Почти в каждой клетке любого животного имеется набор хромосом, несущих информацию об этом животном. Основная часть хромосомы - нить ДНК (молекула дезоксирибонуклеиновой кислоты), которая состоит из четырех видов специальных соединений - нуклеотидов, идущих в определенной последовательности. Нуклеотиды обозначаются буквами A, T, C и G, и именно порядок их следования кодирует все генетические свойства данного организма. Говоря более точно, ДНК определяет, какие химические реакции будут происходить в данной клетке, как она будет развиваться и какие функции выполнять.

Ген — это отрезок цепи ДНК, отвечающий за определенное свойство особи, например за цвет глаз, тип волос, цвет кожи и т. д. Вся совокупность генетических признаков человека кодируется посредством примерно 60 тыс. генов, суммарная длина которых составляет более 90 млн. нуклеотидов.

Различают два вида клеток: половые (такие, как сперматозоид и яйцеклетка) и соматические. В каждой соматической клетке человека содержится 46 хромосом. Эти 46 хромосом — на самом деле 23 пары, причем в каждой паре одна из хромосом получена от отца, а вторая — от матери. Парные хромосомы отвечают за одни и те же признаки — например, отцовская хромосома может содержать ген черного цвета глаз, а парная ей материнская — ген голубоглазости. Существуют определенные законы, управляющие участием тех или иных генов в развитии особи. В частности, в нашем примере потомок будет черноглазым, так как ген голубых глаз является “слабым” (рецессивным) и подавляется геном любого другого цвета.

В половых клетках хромосом только 23, и они непарные. При оплодотворении происходит слияние мужской и женской половых клеток и образуется клетка зародыша, содержащая как раз 46 хромосом. Какие свойства потомок получит от отца, а какие — от матери?

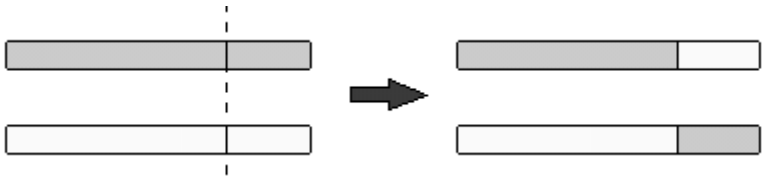

Рис.1 Условная схема кроссовера.

Это зависит от того, какие именно половые клетки участвовали в оплодотворении. Дело в том, что процесс выработки половых клеток (так называемый мейоз) в организме подвержен случайностям, благодаря которым потомки все же во многом отличаются от своих родителей. При мейозе, в частности, происходит следующее: парные хромосомы соматической клетки сближаются вплотную, затем их нити ДНК разрываются в нескольких случайных местах и хромосомы обмениваются своими частями (рис. 1).

Этот процесс обеспечивает появление новых вариантов хромосом и носит название “кроссовер”. Каждая из вновь появившихся хромосом окажется затем внутри одной из половых клеток, и ее генетическая информация может реализоваться в потомках данной особи.

Второй важный фактор, влияющий на наследственность, — это мутации, которые выражаются в изменении некоторых участков ДНК. Мутации также случайны и могут быть вызваны различными внешними факторами, такими, как радиоактивное облучение. Если мутация произошла в половой клетке, то измененный ген может передаться потомку и проявиться в виде наследственной болезни либо в других новых свойствах потомка. Считается, что именно мутации являются причиной появления новых биологических видов, а кроссовер определяет уже изменчивость внутри вида (например, генетические различия между людьми).

Как уже было отмечено выше, эволюция — это процесс постоянной оптимизации биологических видов. Естественный отбор гарантирует, что наиболее приспособленные особи дадут достаточно большое потомство, а благодаря генетическому наследованию мы можем быть уверены, что часть этого потомства не только сохранит высокую приспособленность родителей, но будет обладать и некоторыми новыми свойствами. Если эти новые свойства окажутся полезными, то с большой вероятностью они перейдут и в следующее поколение. Таким образом, происходит накопление полезных качеств и постепенное повышение приспособляемости биологического вида в целом. Зная, как решается задача оптимизации видов в природе, мы теперь применим похожий метод для решения различных реальных задач.

Введем обозначения и приведем несколько классических примеров. Как правило, в задаче оптимизации мы можем управлять несколькими параметрами (обозначим их значения через x_1, x_2, ..., x_n, а нашей целью является максимизация (или минимизация) некоторой функции, $f(x_1, x_2, ..., x_n)$, зависящей от этих параметров. Функция f называется целевой функцией.

Например, если требуется максимизировать целевую функцию “доход компании”, то управляемыми параметрами будут число сотрудников компании, объем производства, затраты на рекламу, цены на конечные продукты и т. д. Важно отметить, что эти параметры связаны между собой — в частности, при уменьшении числа сотрудников скорее всего упадет и объем производства.

Для такого рода задач было разработано несколько методов решений. В случае, если целевая функция достаточно гладкая и имеет только один локальный максимум (унимодальна), то оптимальное решение можно получить методом градиентного спуска. Идея этого метода состоит в том, что оптимальное решение получается итерациями. Берется случайная начальная точка, а затем в цикле происходит сдвиг этой точки на малый шаг, причем шаг делается в том направлении, в котором целевая функция растет быстрее всего. Недостатком градиентного алгоритма являются слишком высокие требования к функции — на практике унимодальность встречается крайне редко, а для неправильной функции градиентный метод часто приводит к неоптимальному ответу. Аналогичные проблемы возникают и с применением других математических методов. Во многих важных задачах параметры могут принимать лишь определенные значения, причем во всех остальных точках целевая функция не определена. Конечно, в этом случае не может быть и речи о ее гладкости и требуются принципиально другие подходы.

Представим себе искусственный мир, населенный множеством существ (особей), причем каждое существо — это некоторое решение нашей задачи. Будем считать особь тем более приспособленной, чем лучше соответствующее решение (чем большее значение целевой функции оно дает). Тогда задача максимизации целевой функции сводится к поиску наиболее приспособленного существа. Конечно, мы не можем поселить в наш виртуальный мир все существа сразу, так как их очень много. Вместо этого мы будем рассматривать много поколений, сменяющих друг друга. Теперь, если мы сумеем ввести в действие естественный отбор и генетическое наследование, то полученный мир будет подчиняться законам эволюции. Заметим, что, в соответствии с нашим определением приспособленности, целью этой искусственной эволюции будет как раз создание наилучших решений. Очевидно, эволюция — бесконечный процесс, в ходе которого приспособленность особей постепенно повышается. Принудительно остановив этот процесс через достаточно долгое время после его начала и выбрав наиболее приспособленную особь в текущем поколении, мы получим не абсолютно точный, но близкий к оптимальному ответ. Такова, вкратце, идея генетического алгоритма. Перейдем теперь к точным определениям и опишем работу генетического алгоритма более детально.

Для того чтобы говорить о генетическом наследовании, нужно снабдить наши существа хромосомами. В генетическом алгоритме хромосома — это некоторый числовой вектор, соответствующий подбираемому параметру, а набор хромосом данной особи определяет решение задачи. Какие именно

векторы следует рассматривать в конкретной задаче, решает сам пользователь. Каждая из позиций вектора хромосомы называется ген.

Определим теперь понятия, соответствующие мутации и кроссоверу в генетическом алгоритме.

Мутация — это преобразование хромосомы, случайно изменяющее одну или несколько ее позиций (генов). Наиболее распространенный вид мутаций — случайное изменение только одного из генов хромосомы.

Кроссовер (в литературе по ГА также употребляется название ***кроссинговер*** и ***скрещивание***) — это операция, при которой из двух хромосом порождается одна или несколько новых хромосом. В простейшем случае кроссовер в генетическом алгоритме реализуется так же, как и в биологии (см. рис. 2.1). При этом хромосомы разрезаются в случайной точке и обмениваются частями между собой. Например, если хромосомы (1, 2, 3, 4, 5) и (0, 0, 0, 0, 0) разрезать между третьим и четвертым генами и обменять их части, то получатся потомки (1, 2, 3, 0, 0) и (0, 0, 0, 4, 5).

Блок-схема генетического алгоритма изображена на рис. 2.

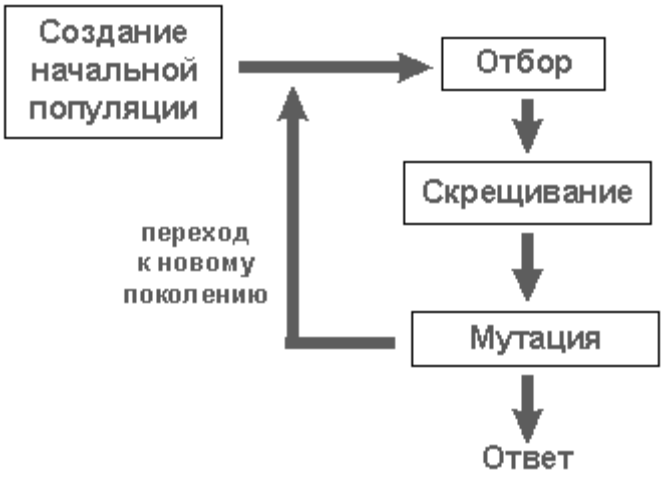

Рис. 2. Блок-схема генетического алгоритма.

Вначале генерируется начальная популяция особей (индивидуумов), т.е. некоторый набор решений задачи. Как правило, это делается случайным образом. Затем мы должны смоделировать размножение внутри этой популяции. Для этого случайно отбираются несколько пар индивидуумов, производится скрещивание между хромосомами в каждой паре, а полученные новые хромосомы помещаются в популяцию нового поколения. В генетическом алгоритме сохраняется основной принцип естественного отбора — чем приспособленнее индивидуум (чем больше соответствующее ему значение целевой функции), тем с большей вероятностью он будет участвовать в скрещивании. Теперь моделируются мутации — в нескольких случайно выбранных особях нового поколения изменяются некоторые гены. Затем старая популяция частично или полностью уничтожается и мы переходим к рассмотрению следующего поколения. Популяция следующего поколения в большинстве реализаций ГА содержит столько же особей, сколько начальная, но в силу отбора приспособленность в ней в среднем

выше. Теперь описанные процессы отбора, скрещивания и мутации повторяются уже для этой популяции и т. д.

В каждом следующем поколении мы будем наблюдать возникновение совершенно новых решений нашей задачи. Среди них будут как плохие, так и хорошие, но благодаря отбору число хороших решений будет возрастать. Заметим, что в природе не бывает абсолютных гарантий, и даже самый приспособленный тигр может погибнуть от ружейного выстрела, не оставив потомства. Имитируя эволюцию на компьютере, мы можем избегать подобных нежелательных событий и всегда сохранять жизнь лучшему из индивидуумов текущего поколения — такая методика называется “стратегией элитизма”.

Реализация генетических алгоритмов в MathLab

В MathLab возможность использования ГА для вычислений реализована с помощью вкладки Genetic *Algorithm and Direct Sarch Toolbox*, которая расширяет возможности пакета *Optimization Toolbox*, генетическими алгоритмами. Такие алгоритмы чаще всего Такие алгоритмы чаще всего используются в случае, когда искомая целевая функция является разрывной, существенно нелинейной, стохастической и не имеет производных или эти производные являются недостаточно определенными. Работать с генетическими алгоритмами теперь можно в двух тулбоксах.

Собственно генетические алгоритмы относятся к разделу *Genetic Algorithm* и вызываются из командной сроки с помощью *gatool* или *ga*.
Генетические алгоритмы и их комбинации с другими оптимизационными методами можно найти в разделе *Direct Search Toolbox*. Для этого в командной строке необходимо набрать psearchtool.
Рассмотрим первый вариант работы с ГА. Существуют 4 основные функции для работы с алгоритмом:
ga — функция для нахождения минимума целевой функции;
gaoptimget — возвращает параметры используемого генетического алгоритма;
gaoptimset — устанавливает параметры генетического алгоритма;
gatool — открывает окно *Genetic Algorithm Tool*.
Для того, чтобы применить ГА к поставленной функции цели, необходимо в первую очередь записать её в M-file и сохранить в текущей папке.

В.2. Функция ga

Функция ga вызывается в командной строке согласно нижеприведенному синтаксису:
[x fval] = ga(@fitnessfun, nvars, options)
здесь
fitnessfun — имя M-file, содержащего поставленную целевую функцию;
nvars — число независимых переменных в целевой функции;

options — структура, содержащая параметры используемого ГА. Если параметры не изменять, то их значения возьмутся по умолчанию;
Результаты вычислений сохранятся в переменных:
fval — окончательное значение целевой функции;
x — точка, в которой достигнуто оптимальное значение.
Существуют и другие варианты вызова функции ga:
x = ga(fitnessfun, nvars)
x = ga(fitnessfun, nvars, options)
x = ga(problem)
[x, fval] = ga(...)
[x, fval, reason] = ga(...)
[x, fval, reason, output] = ga(...)
[x, fval, reason, output, population] = ga(...)
[x, fval, reason, output, population, scores] = ga(...)
Описание x = ga(fitnessfun, nvars) применяется для решения оптимизационной задачи, fitnessfun — минимизируемая целевая функция и nvars — длина вектора решений x, соответствующего наилучшей особи.
x = ga(fitnessfun, nvars, options) применяется для решения оптимизационной задачи, используя параметры (options) алгоритма.
x = ga(problem) находит минимум задачи, структура которой описывается тремя полями:
fitnessfcn — целевая функция;
nvars — число независимых переменных целевой функции;
options — параметры структуры ГА, задаваемые функцией gaoptimset.
[x, fval] = ga(...) возвращает fval, значение целевой функции по x.
[x, fval, reason] = ga(...) возвращает reason — строку, содержащую параметры остановки алгоритма.
[x, fval, reason, output] = ga(...) возвращает output совокупность сведений о каждом поколении и другой информации о реализации алгоритма. Структура output состоит из следующих полей:
Randstate или (randnstate) — начальное состояние популяции сгенерированное случайными числами. (Различие функций состоит в разных выводах.);
generations — количество вычисляемых поколений;
funccount — количество вычислений функции;
message — параметры остановки алгоритма. Это сообщение выводит несколько аргументов завершения алгоритма.
[x, fval, reason, output, population] = ga(...) возвращает матрицу популяции, строки которой соответствуют особи конечной популяции.
[x, fval, reason, output, population, scores] = ga(...) возвращает расчеты финальной популяции.

Замечание: для всех оптимизационных задач популяция должна быть представлена в виде вещественных чисел. Функция ga не работает для функций с комплексными переменными. Для решения задач включающих оптимизационные числа нужно записать целевую функцию в виде

допустимого вещественного вектора, отделив вещественные и комплексные части.

Пример

```
[x fval, reason] = ga(@rastriginsFcn, 10)
x =
Columns 1 through 7
0.9977   0.9598  0.0085 0.0097 -0.0274 -0.0173 0.9650
Columns 8 through 10
-0.0021 -0.0210  0.0065
fval =
3.7456
reason =
generations
```

Функция gaoptimset

Для настройки генетического алгоритма используется функция gaoptimset. Она позволяет построить ГА комбинируя, операторы по желанию пользователя. Синтаксис данной функции выглядит следующим образом.

Синтаксис

```
options = gaoptimset
gaoptimset
options = gaoptimset('param1',value1,'param2',value2,...)
options = gaoptimset(oldopts,'param1',value1,...)
options = gaoptimset(oldopts,newopts)
```

Описание

options = gaoptimset (здесь аргументы не вводятся) с помощью данной конструкции задается структура ГА, отличная от структуры ГА по умолчанию.

gaoptimset не требует ввода или вывода аргументов. В результате сформирует список параметров и их действительных значений.

options = gaoptimset('param1',value1,'param2',value2,...) генерирует структуру с множеством параметров их значений. Для нескольких неспециальных параметров можно использовать значения по умолчанию.

options = gaoptimset(oldopts,'param1',value1,...) создает копию oldopts, модифицированную выбранными специальными параметрами и их значениями.

options = gaoptimset(oldopts,newopts) комбинирует параметры существующей структуры oldopts, с параметрами новых структур newopts. Некоторые параметры с ненулевыми значениями в newopts могут заменить или быть присвоены старым параметрам в oldopts.

Опции

В таблице приведен список параметров для функции gaoptimset. В фигурных скобках {} берутся значения по умолчанию.

Таблица. Список параметров функции gaoptimset.

Опции	**Описание**	**Значения**
CreationFcn	Используется для создания начальной популяции	{@gacreationuniform}
CrossoverFraction	Вероятность кроссинговера, не включает элитных потомков, полученных при кроссинговере	Положительные числа | {0.8}
CrossoverFcn	Вид кроссинговера	@crossoverheuristic (случайный кроссинговер) {@crossoverscattered} @crossoverintermediate @crossoversinglepoint (одноточечный кроссинговер) @crossovertwopoint (двуточечный кроссинговер)
EliteCount	Положительное целое число, определяющее количество особей текущей популяции, которые будут скопированы в новое поколение	Положительное целое | {2}
FitnessLimit	Число. Если функция пригодности достигнет значения FitnessLimit, то алгоритм остановится	Число | {-Inf}
FitnessScalingFcn	Устанавливается, если значениями функции пригодности являются числа	@fitscalinggoldberg {@fitscalingrank} @fitscalingprop @fitscalingtop
Generations	Положительное число, определяющее максимальное число итераций алгоритма	Положительное целое |{100}

Опции	**Описание**	**Значения**
PopInitRange	Матрица или вектор, определяющие особи в начальной популяции	Матрица или вектор | [0;1]
PopulationType	Строковое выражение, описывающее тип данных в популяции	'bitstring' | 'custom' | {'doubleVector'}
HybridFcn	Устанавливает дальнейшую оптимизацию по завершению работы ga	Установочная функция | {[]}
InitialPopulation	Начальная популяция	Положительные числа | {[]}
InitialScores	Начальные точки	Вектор-столбец | {[]}
MigrationDirection	Направление миграции	'both' | {'forward'}
MigrationFraction	Число между 0 и 1, соответствующее доли мигрирующих особей для каждой подпопуляции	Число | {0.2}
MigrationInterval	Положительное целое соответствующее количеству поколений между миграциями	Положительное целое | {20}
MutationFcn	Устанавливается в случае мутации потомков	@mutationuniform {@mutationgaussian}
OutputFcns	Массив, содержащий функции, которые ГА вызывал каждую итерацию	массив | {[]}
OutputInterval	Положительное целое число поколений между последовательными вызовами функции вывода	Положительное целое | {1}

Опции	Описание	Значения
PlotFcns	Массив для сохранения результатов, полученных в ходе вычислений алгоритма. В дальнейшем используется для построения графика	@gaplotbestf @gaplotbestgenome @gaplotdistance @gaplotexpectation @gaplotgeneology @gaplotselection @gaplotrange @gaplotscorediversity @gaplotscores @gaplotstopping | {[]}
PlotInterval	Положительное целое число поколений между последовательными вызовами функции plot	Положительное целое | {1}
PopulationSize	Размер популяции	Положительное целое | {20}
SelectionFcn	Устанавливает функцию отбора родителей для кроссинговера и мутации потомков	@selectiongoldberg @selectionrandom {@selectionstochunif} @selectionroulette @selectiontournament
StallGenLimit	Положительное целое. Алгоритм остановится, если за StallGenLimit последовательных итераций не произойдет улучшение функции цели	Положительное целое | {50}
StallTimeLimit	Положительное целое. Алгоритм остановится, если за StallTimeLimit секунд не произойдет улучшение функции цели	Положительное целое | {20}
TimeLimit	Положительное число. Алгоритм остановится после TimeLimit секунд с момента начала работы	Положительное число | {30}
Vectorized	Строка, определяющая векторизацию функции цели	'on' | {'off'}

Например:

1. Функция gaoptimget

Функция gaoptimget обладает следующим синтаксисом:

val = gaoptimget(options, 'name');

2. Функция gatool.

3. Другой наиболее наглядный способ работы с ГА заключается в вызове окна с помощью функции gatool.

Диалоговое окно представлено на рис. 3.

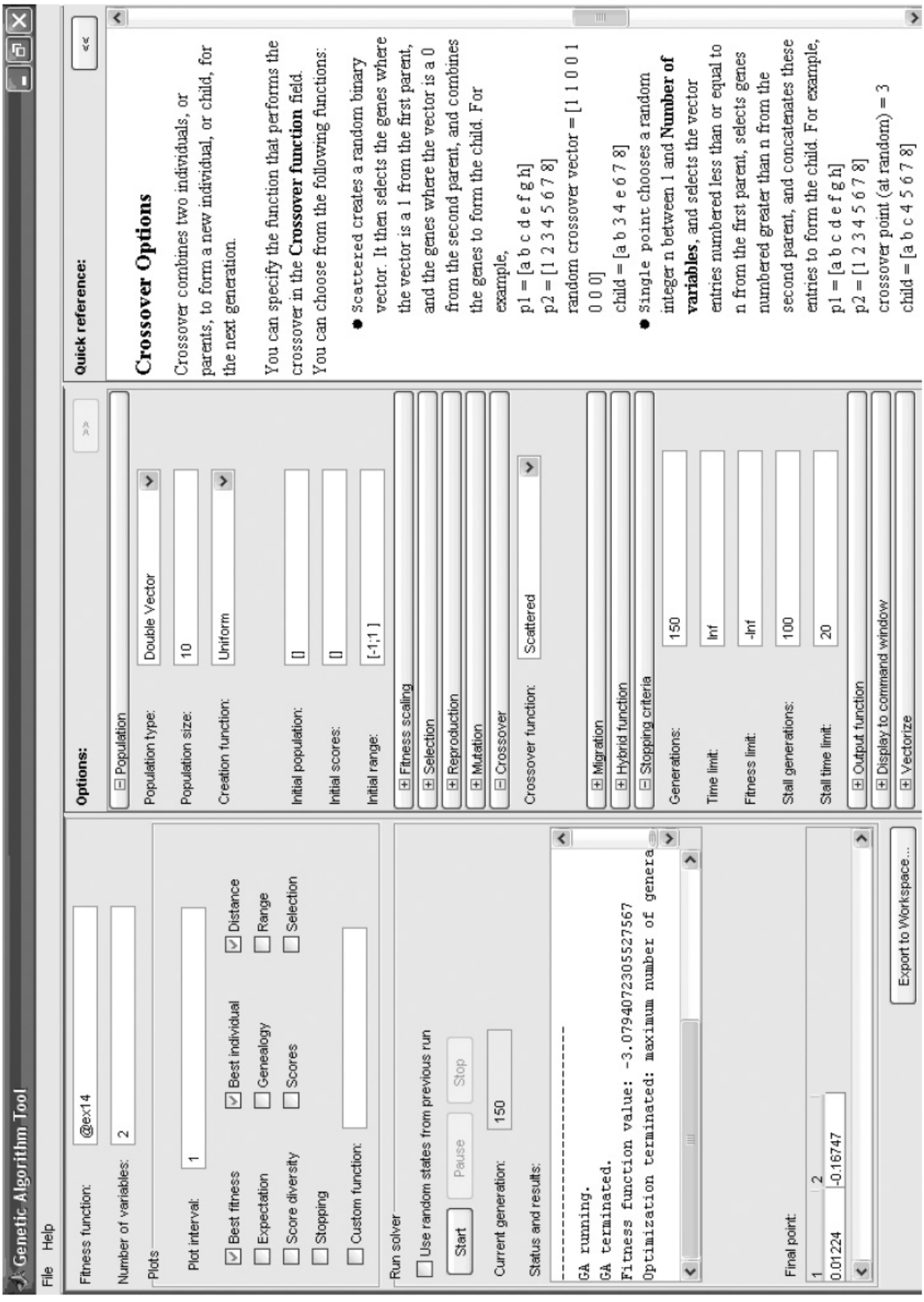

Рис. 3. Диалоговое окно по генетическим алгоритмам.

Как видно из рисунка, окно сопровождено справкой по всем компонентам, и работа пользователя заключается в виде установки параметров и нажатии кнопки «start». Результат будет тем же, что и в случае последовательного применения функций gaoptimset и ga. В разделе plots можно выбрать переменные, изменение которых будет отображаться графически.

Векторизация целевой функции

В диалоговом окне ГА предусмотрена векторизация функций, благодаря чемувычисления происходят заметно быстрее. Смысл данного метода заключается в том, что в качестве параметров функции выступают вектора, тогда для текущей популяции целевая функция будет вызываться лишь один раз, вычисляя пригодности всех особей. Например, рассмотрим функцию $f(x_1, x_2) = x_{21} - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2$.

Запишем для нej M-file, используя следующий код:

```
z =x(:,1).\ 2 - 2*x(:,1).*x(:,2) + 6*x(:,1) +x(:,2).\ 2-6*x(:,2);
```

Здесь x(:, 1) представляет собой вектор, а .\ и .* — операции поэлементного соответственно возведения в степень и умножения.

В окне тулбокса в списке Vectorize option следует установить значение On.

Убедиться в эффективности векторизации можно на примере функции Растригина. В командной строке введем следующее выражение:

```
tic;ga(@rastriginsfcn,20);toc
```

В результате можно узнать время, затраченное на вычисления:

```
elapsed_time =
4.3660
73
```

Проделаем тоже самое, но используя векторизацию функции Растригина:

```
options=gaoptimset('Vectorize','on');
tic;ga(@rastriginsfcn,20,options);toc \
```

Узнаем время затраченное на векторизацию:

```
elapsed_time = 0.581
```

Как видно метод векторизации работает на много быстрее.

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Перед выполнением лабораторных работ необходимо внимательно прочитать и уяснить решение первой задачи лабораторной работы.

Задание 1

Минимизировать функцию одной переменной

$$f(x) = 8x - 16 - 12\sqrt[3]{(x+4)^2}$$

Решение

Напишем M-file для данной функции и сохраним его в текущей папке под именем ex2.m.

```
function y = ex2(x)
y=-12*(x+4)\ (2/3)+8*x-16;
```

Вызовем окно тулбокса с помощью *gatool*.

В поле *fitness function* введем имя целевой функции @ex2

Установим значения параметров ГА: количество особей в популяции = 10, количество поколений = 100 (в окне критерия остановки алгоритма), начальный отрезок = [-4; 1]. В разделе *plots* установим флажки для *best fitness, best individual, distance*. Щелкнем по кнопке *start*.

В результате завершения процесса в окне *final point* появится значение переменной x, соответствующее минимуму функции, а в окне *status and result* можно увидеть найденное минимальное значение целевой функции.

Для данной задачи результаты получились следующие минимум функции достигается в точке x = -2.9972 и f(-2.9972) = -51.99998959105959.

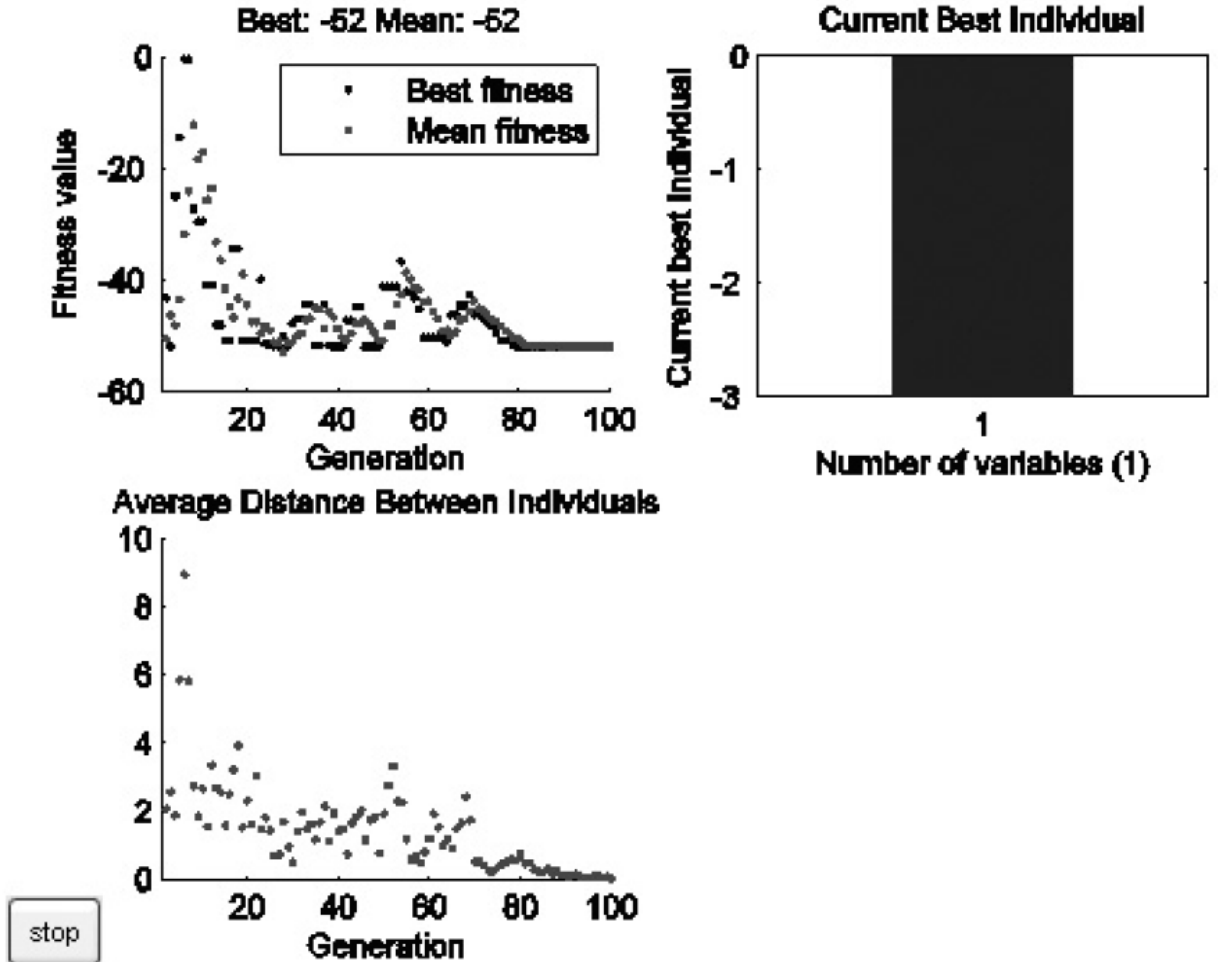

Рис.4. Графический анализ решения.

Первый рисунок (рис. 4) отображает изменение значение целевой функции. Видно, что, начиная с 80 популяции, алгоритм сошелся к решению. На втором рисунке изображена наилучшая особь. Третий рисунок

соответствует изменению расстояния между особями в поколениях. Особи становятся одинаковыми (хеммингово расстояние = 0) в последних 18 поколениях. ГА нужно запустить несколько раз, а потом выбрать оптимальное решение. Это связано с тем, что начальная популяция формируется с использованием генератора случайных чисел.

Убедиться в правильности решения можно, построив график функции (рис. 5).

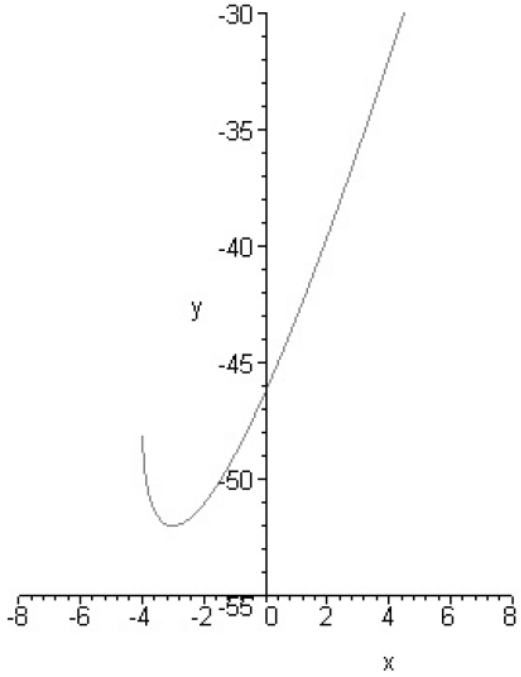

Рис. 5. График функции

То же самое можно было бы получить, используя функции *gaoptimset* и *ga*. Чтобы посмотреть M-File выберете в меню «File» окна «Genetic Algoritm Tool» команду «Generate M-file», сохраните файл под другим именем и просмотрите код. Для данной задачи получили:

```
function [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ex2q
% This is an auto generated M file to do optimization
% with the Genetic Algorithm and
% Direct Search Toolbox. Use GAOPTIMSET for default
% GA options structure.
% Fitness function
fitnessFunction = @ex2;
% Number of Variables
nvars = 1 ;
% Start with default options
options = gaoptimset;
% Modify some parameters
options = gaoptimset(options,'PopInitRange',[-4 ; -1 ]);
options = gaoptimset(options,'PopulationSize',10);
```

```
options = gaoptimset(options,'MutationFcn',
{@mutationgaussian 1 1});
options = gaoptimset(options,'Display','off');
options = gaoptimset(options,'PlotFcns', {@gaplotbestf
@gaplotbestindiv @gaplotdistance });
% Run GA
[X,FVAL,REASON,OUTPUT,POPULATION,SCORES] =
ga(fitnessFunction,nvars,options);
```

Задание 2

Максимизировать функцию двух переменных:

$$z(x, y) = \exp(-x^2 - y^2) + \sin(x + y).$$

Решение

В диалоговом окне ГА можно решить только задачи минимизации. Для нахождения максимума функции *f(x)* следует минимизировать функцию *-f(x)*. Этообъясняется тем, что точка минимума *-f(x)* является некоторой точкой *f(x)*, в которой достигается максимум.

Напишем M-file для функции *z(x) = -f(x)* и сохраним его в текущей папке под именем ex13.m:

```
function z = ex13(x)
z=-(exp(-x(1)\ 2-x(2)\ 2)+sin(x(1)+x(2)));
```

Вызовем диалоговое окно с помощью *gatool*.

В поле *fitness function* введем имя целевой функции @ex13.

Установим значения параметров ГА: количество переменных = 2, количество особей в популяции = 10, количество поколений = 100 (в окне критерия остановки алгоритма), начальный отрезок = [-1; 3]. Для построения графиков в разделе *plots* установим флажки для *best fitness, best individual, distance*. Щелкнем по кнопке *start*.

В результате завершения процесса в окне *final point* появится значение переменной *x*, соответствующее минимуму функции, а в окне *status and result* можно увидеть найденное минимальное значение целевой функции *z(x)*.

Для данной задачи результаты получились следующие: максимум функции достигается в точке $x = 0.46419$, $y = 0.42406$ и $f(0.46419; 0.42406) = 1.449$.

Варианты задания 2

1. Найти максимум функции $y(x) = 3\sqrt[3]{(x+4)^2} - 2x - 8$.
2. Найти минимум функции $z(x, y) = (y - 3)\exp(-x^2 - y^2)$.
3. Построить тестовую функцию Эккли для двух переменных.

Варианты задания 3

1. Найти максимум функции $y(x)=\dfrac{6\sqrt[3]{6(x-3)^2}}{(x-1)^2+8}$.
2. Найти минимум и максимум функции $z(x,y)=x\exp\left(-x^2-y^2\right)$.
3. Построить тестовую функцию Михалевича для двух переменных.

Также можно сконструировать другие ГА, моделируя операторы (выбор родительских пар, кроссинговер, мутация, миграция, отбор особей в новую популяцию, критерии завершения алгоритма) и параметры алгоритма. Дальнейшее изучение оптимизации с помощью ГА можно продолжить, рассматривая *Direct Search Toolbox*. Для вызова окна можно в командной строке просто прописать *psearchtool*.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение генетического алгоритма.
2. Каковы основные термины, используемые при реализации ГА и взятые из биологии.
3. Каковы основные задачи, решаемые с помощью ГА?
4. Как реализовать ГА в среде MathLab?
5. В чем особенности использования инструментов MathLab при решении задач оптимизации с помощью ГА?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Т.В. Панченко. Генетические алгоритмы. Учебное пособие. Астрахань: Изд. дом «Астраханский университет». 2007 – 87 с.