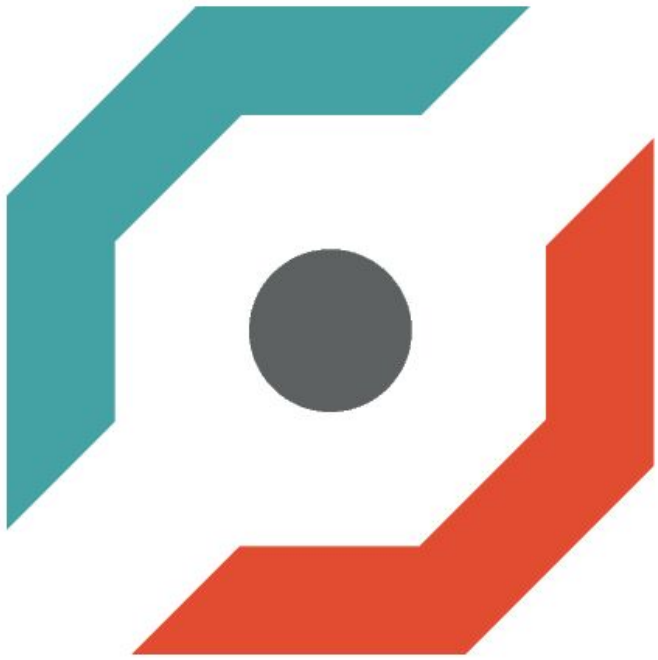

ИНБИКСТ

Решение систем линейных уравнений генетическими алгоритмами

15.12.2017

Леонид Куликов
МІРТ, ИНБИКСТ, гр. 6103
Преподаватель: Федотов В. А.
Москва

Преамбула

Генетические алгоритмы - адаптивные методы поиска, которые в последнее время часто используются для решения задач функциональной оптимизации. Они основаны на генетических процессах биологических организмов. В их основе лежит естественный отбор. Таким образом если правильно задать функцию отбора, то можно за довольно быстрое время найти, к примеру, решение системы линейных уравнений вида Ax=b.

Описание алгоритма

В качестве “ядра” будущей программы я взял классическую вариацию генетического алгоритма (http://algolist.manual.ru) с минимальными модификациями.

Суть

1. Определение различных констант, о которых будет рассказано ниже.
2. Создание начальной популяции X (членом этой популяции является x - искомое решение)
3. Вычисление коэффициентов выживания для каждого члена популяции (в качестве такой функции я использую Survival[x] = [1/|Ax-b|]/SUM, где SUM = Σ1/Survival[X]). Выбор именно такой функции обусловлен тем, что чем ближе решение Ax к b, тем больше коэффициент выживания.
4. Далее производим N/2 скрещиваний в соответствии с Survival[x] (N-размер популяции, приоритет при скрещивании отдается парам с большими значениями коэффициентов выживания). Потомки занимают места родителей.
5. Мутирование популяции. Мутация случайным образом заменяет значения x_i у всех членов популяции.
6. Проверка на удовлетворительное решение (решение с погрешностью).
7. Повтор начиная с 3) до тех пор пока не найдется удовлетворительное решение или не исчерпается лимит поколений у популяции.

Назначения констант

N_GEN

Определяет сколько итераций с изменением членов популяции можно произвести (Лимит поколений).

N

Число членов популяции.

MIN, MAX

Пределы для x_i.

MUT

Коэффициент мутации в процентах

EPS

Допустимая погрешность для решения.

Описание работы функций (кроме main)

Mutation

Случайным образом заменяет x_i для данного члена популяции.

Mult_AiX

Возвращает A_i*x - b_i, где A_i -- i-ая строка.

Random

Возвращает рандомное вещественное число в пределах от MIN до MAX.

Cross_prob

Выбирает пару для скрещивания.

Пара выбирается на основе коэффициентов выживания. Для этого строится специальный вектор **dist**, в который помещаются индексы каждого члена популяции в количестве равном их коэффициенту выживания, выраженному в процентах. Далее выбирается случайный элемент в этом массиве и кладется в переменную **p1**, то есть член популяции с индексом p1 становится участником скрещивания. Затем в векторе **dist** удаляются все элементы с индексом p1 (ибо считаем, что для скрещивания

необходимы 2 разных члена популяции). Затем таким же образом выбирается **p2**. Позднее вычисляется **p** - среднее-арифметическое их коэффициентов выживания в процентах, чтобы задать переменную **crossover**. В случае если она принимает значение **true**, то выбранная пара переходит к стадии скрещивания. Для этого в строку **wheel** помещаем единички в количестве, равном **p**, остальные позиции заполняем нулями (всего позиций в wheel = 100).

Описание работы main

После объявления большей части переменных (я постарался их назвать так, чтобы по названию было понятно, зачем они, но на всякий случай также добавил краткую справку в комментах) идет **switch**. Он необходим для организации пользовательского меню. Непосредственно дальше, начинается организация алгоритма, упомянутого в разделе “Описание алгоритма”. Для определения коэффициентов выживания я использую вложенный цикл, так как подаю в функцию Mult_AiX только строки. Таким образом Survival[x] считается как сумма по i от 0 до dim A_ix-bi. После подсчета Survival[x], он добавляется в sum как 1/Survival[x]. Позже (после подсчета всех Survival[x]) Survival[x] приравнивается к 1/Survival[x]/sum.

Далее происходит скрещивание N/2 раз. После выбора родителей, происходит проверка переменной **crossover**, в случае если она принимает значение true, то скрещивание успешно. Для скрещивания вводятся дополнительные переменные: **div_cross** - которая отмечает точку среди элементов x_i у родителя (таким образом разделяя x на две половины) и **which_size_change** - которая отвечает за то, какая половина будет претерпевать изменения. Далее происходит обмен значений x_i согласно их индексам.

Обновленное поколение переходит, далее, к мутации. По сути, это обычный цикл, который проходит по всем членам популяции и заменяет их x_i с вероятностью **MUT**.

После происходит проверка на решение. Для случая когда решение с заданной погрешностью не нашлось вводятся переменные **suit_sol** - индекс лучше решения в векторе популяции и **suit_defect** - минимальная погрешность полученная в ходе работы программы.

Эпилог

На текущий момент программа общается с пользователем посредством консоли. Может искать решения с погрешность <=1 для всех x_i за разумное время. Спасибо за прочтение!