UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
COMPUTAÇÃO NATURAL

Trabalho Prático 1

Programação Genética em Regressão Simbólica

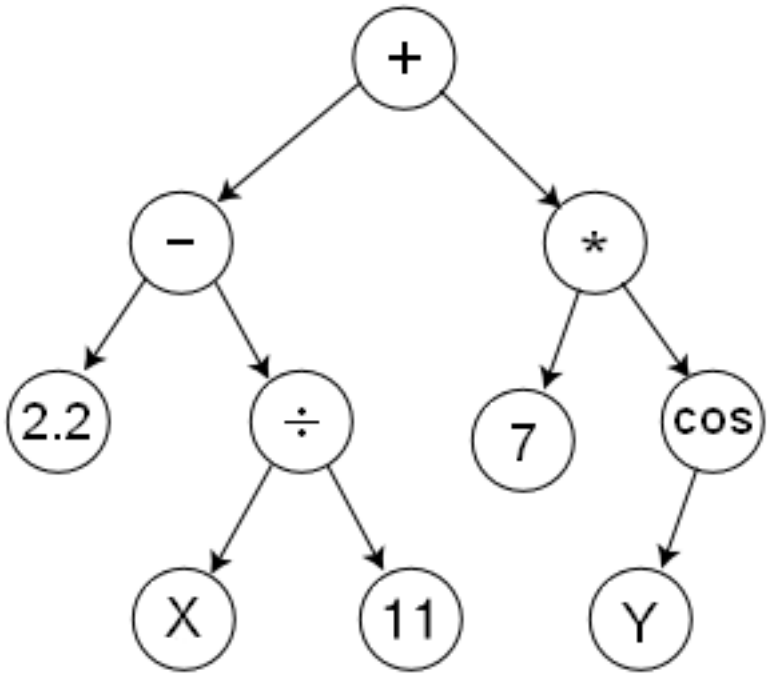
Aluno:
Pedro Henrique Ramos Costa (2011049436)

Outubro de 2015, Belo Horizonte.

1. INTRODUÇÃO

Programação genética é uma metodologia de programação baseado em algoritmos evolucionários e bioinspirados para encontrar programas de computador que executem bem uma determinada tarefa. Essencialmente, PG é um conjunto de instruções e uma função de *fitness* que mede quão bem o programa executou aquela tarefa. É uma especialização de Algoritmos Genéticos (AG).

Tradicionalmente, PG evolui indivíduos representados em memória por estruturas de árvores. Árvores podem ser facilmente avaliadas de forma recursiva. Cada nó da árvore contém um operador (uma função) e cada folha pode conter um terminal que seja uma variável ou uma constante, o que torna expressões matemáticas fáceis de serem representadas, evoluídas e avaliadas. Dessa forma, PG tradicionalmente incita o uso de linguagens que encorporam naturalmente árvores (como Lisp, e algumas linguagens funcionais).



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

Os principais operadores utilizados em algoritmos evolucionários também são utilizados em PG, e são os operadores de cruzamento (**crossover**) e mutação (**mutation**).

Crossover:

O operador de crossover é aplicado sobre um indivíduo simplesmente trocando um de seus nós com o nó de um outro indivíduo numa mesma população. Com uma representação em árvores, substituir um nó significa substituir todo o galho (ramo) da árvore. Isso adiciona grande efetividade ao operador de crossover, pois provém diversidade. O resultado dos crossovers costuma ser bem diferente dos pais iniciais.

Mutação:

Mutação afeta um indivíduo somente. Na mutação, um nó sofre uma mutação, ou seja, muda sua informação. Um nó pode simplesmente virar um terminal, e dessa forma o ramo que dele partia deixará de existir; ou pode mudar o seu valor, caso seja uma função; ou pode simplesmente mudar seu valor caso seja um terminal (variável ou constante).

PG aplicada à Regressão Simbólica

Regressão simbólica é um tipo de análise de regressão que procura, em um espaço de expressões matemáticas, aquele modelo que melhor se adequa a um dado conjunto de dados, ambos em termos de simplicidade e acurácia. Não há nenhum modelo inicial. Pelo contrário, parte-se de uma expressão constituída de forma aleatória, até que novas equações são formadas por recombinação das expressões geradas inicialmente, utilizando programação genética.

Por não exigir um modelo específico a ser seguido, a regressão simbólica não é afetado por nenhum ponto de vista humano. Ela procura descobrir as relações intrínsecas no conjunto de dados inicial, permitindo que padrões nos

dados revelem eles mesmos os modelos apropriados, ao invés de impor um modelo estrutural que seja matematicamente proposto a partir de uma perspectiva humana. A função de fitness que rege a evolução dos modelos (indivíduos) leva em consideração métricas de erros e outras medidas mais complexas que se certificam de tornar um indivíduo compreensível por uma perspectiva humana.

2. OBJETIVO

Este trabalho tem como objetivo desenvolver um programa genético para realizar regressão simbólica sobre um determinado conjunto de coordenadas, que constituem o dataset. O foco do trabalho é desenvolver um PG cujos parâmetros melhor encontrem uma função para descrever um conjunto de dados com 50 coordenadas e 6 variáveis. Um outro conjunto de dados com funções já conhecidas matematicamente também foi utilizado para testes.

3. DECISÕES DE IMPLEMENTAÇÃO

O código foi desenvolvido na linguagem **Python**. Os indivíduos foram implementados sob paradigma de Programação Orientada à Objetos, enquanto que o Programa Genético foi implementado proceduralmente.

3.1. Representação do Indivíduo

O indivíduo foi representado por uma árvore de expressão, como já descrito na introdução desde relatório. Testes iniciais foram realizados com seis operadores para os nós de função: (Adição, Subtração, Multiplicação, Divisão, Seno e Cosseno). Porém, testes mostraram que indivíduos com melhor fitness eram mais facilmente encontrados quanto apenas as 4 primeiras funções (+, -, *, \ ) estavam presentes. A definição de quais funções serão utilizadas para gerar os indivíduos é feita de forma *hardcoded*, ou seja, dentro de um dos módulos que constituem o indivíduo.

O genótipo do indivíduo, dessa forma, constitui-se por uma árvore de expressão. Os nós dessa árvore de expressão podem assumir diversos valores. Cada nó é definido pelo seu tipo. Existem três tipos possíveis,

Função: nesse caso o nó pode assumir valores inteiros de 0 a N-1, em que N é o número de funções sendo utilizadas. Este nó deve obrigatoriamente ter dois filhos, pois ele representa uma operação binária.

Variável: nesse caso o nó pode assumir valores inteiros de 0 a N-1, em que N representa o número de variáveis na expressão (X1, X2, ..., Xn). É um nó terminal, ou seja, uma folha, portanto não possui filhos.

Constante: nesse caso o nó pode assumir quaisquer valores reais entre 0 a 1, gerados pelo gerador de números randômicos da biblioteca padrão do Python (random), e definidos por uma seed que representa sempre o número da execução em andamento. Também é um terminal e não possui filhos.

A árvore de expressão tem os parâmetros que definem suas dimensões, como profundidade mínima e máxima, definidos no código do PG, de forma fácil de ser configurada (como constantes globais).

3.2. População

A população inicial é gerada a partir de um método chamado *gen_population(size=initial_size)*. Este método gera uma população pela metodologia ramped half-half. Metade do número de indivíduos sendo gerados serão árvores FULL, ou seja, terão todos os terminais para uma árvore de tamanho máximo definido. A outra metade será gerada pelo método GROW, que gera uma árvore aleatória e sem forma definida, entre os tamanhos mínimo e máximo definidos.

Para a evolução das gerações, utilizou-se elitismo (o melhor indivíduo da geração anterior sempre está presente na população da nova geração).

3.3. Operadores Utilizados

Os operadores utilizados foram o de crossover e o de mutação. As probabilidades de ambos acontecerem são definidas também em variáveis globais no PG. Para a escolha dos pais, utilizou-se o método de **torneio**. O tamanho do torneio também é um parâmetro de fácil configuração.

O Crossover funciona da seguinte maneira: há a tentativa de N/2 cruzamentos, em que N é o tamanho da população. Caso o cruzamento ocorra, os filhos são avaliados e a família (Pais e filhos, 4 indivíduos) é ordenada em relação a sua fitness. Apenas os DOIS melhores indivíduos da família são selecionados para constituir a nova população. Caso a nova população não tenha completado o tamanho exigido da população (exemplo: uma população de 100 indivíduos, houve apenas 45 cruzamentos, portanto apenas 90 indivíduos foram selecionados para iniciarem a nova população, faltando 10 para completar o tamanho padrão da população de 100), então o restante dos indivíduos serão gerados via ramped half-half, da mesma forma que a população inicial foi gerada. Isso promove diversidade.

3.4. Fitness

A Fitness dos indivíduos foi determinada como o erro absoluto entre as coordenadas e o resultado avaliado para a expressão (indivíduo) dadas todas as coordenadas. Mais especificamente,

$$f(Ind) = \sum_{\{\langle x,y\rangle\}} \|\textsc{Eval}(Ind, x) - y\|,$$

Em que IND é o indivíduo sendo avaliado, Eval(Ind, x) avalia o indivíduo dada a variável x, e {<x, y>} é o conjunto de entrada fornecido, e y é a saída correta para a função, para o caso em que cada coordenada tenha apenas 2 pontos.

O indivíduo tem também um atributo de fitness, que é definido durante sua avaliação, e que pode ser alterado via parâmetro. A função fitness que avalia o indivíduo deve receber um conjunto de pontos (coordenadas), somente, e pode ser passada por parâmetro para o indivíduo. Dessa forma a Fitness se torna um

parâmetro fácil de ser alterado, e novas funções fitness podem ser adicionadas ao código.

4. EXPERIMENTOS

Para os experimentos, foram testadas inúmeras configurações dos parâmetros do PG, como tamanho do indivíduo mínimo e máximo, número de operações matemáticas disponíveis, número de gerações, tamanho do torneio, probabilidade de mutação e crossover, tamanho da população, número de gerações. Por limitações de TEMPO, observou-se que para populações acima de 500 e para gerações acima de 100, uma simples execução do algoritmo, com indíviduos de tamanho máximo 6 (profundidade da árvore) se tornou inviável (aproximadamente 1 hora por execução). Isso ocorre provavelmente devido ao **bloating**, que, em situações em que os indivíduos sejam muito grandes (e consequentemente os introns também são grandes), o bloating cresce de forma muito rápida e o algorimo fica lento. Portanto definiu-se alguns parâmetros imutáveis. São eles:

Profundidade mínima da árvore	1
Profundidade máxima da árvore	5
Tamanho da População	500
Número de Gerações	100
Número de indivíduos de "Elite"	1
Número de execuções por seed	30

Outros parâmetros foram variados e analisados pelos testes:

Probabilidade de Crossover
Probabilidade de Mutação
Tamanho do Torneio

Para que algumas configurações específicas fossem definidas, utilizou-se o dataset SR_div.txt e SR_div_noise.txt. Após mais de 300 execuções sobre estes datasets, e muitas configurações diferentes (todos os resultados para os testes sobre esses dois datasets estão no diretório exec_logs) chegou-se a 6 configurações específicas nas quais o algoritmo deveria ser rodado 30 vezes em cada, sobre o dataset principal (yacht_hydrodynamics.txt).

	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6
Probabilidade de Crossover	0.9	0.6	0.3	0.9	0.6	0.3
Probabilidade de Mutação	0.05	0.3	0.9	0.05	0.3	0.9
Tamanho do Torneio	20	20	20	2	2	2

Para chegar a essas configurações, os parâmetros foram mudados e avaliados e diversas execuções foram realizadas sobre o dataset SR_div.txt.

Observou-se que para todas as seeds acima os resultados foram muito parecidos. Os melhores indivíduos encontrados tiveram fitness média na ordem de 10^{-2}. Essas seeds foram escolhidas pois, não somente deram os melhores indivíduos, mas também proveram os melhores resultados quando aplicados sobre o dataset com distorções (noise).

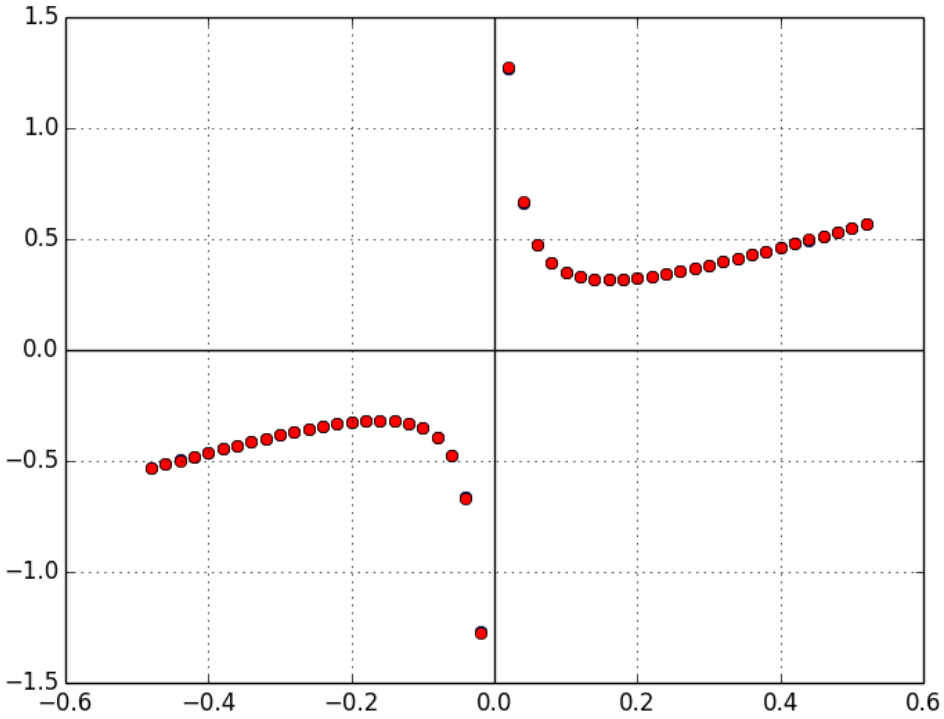
O melhores indivíduos observados no SR_div difere pouco dos melhores indivíduos observado em SR_div_noise. Um exemplo de comparação de um dos indivíduos encontrados para ambos os casos:

Legenda:

Azul: dataset original

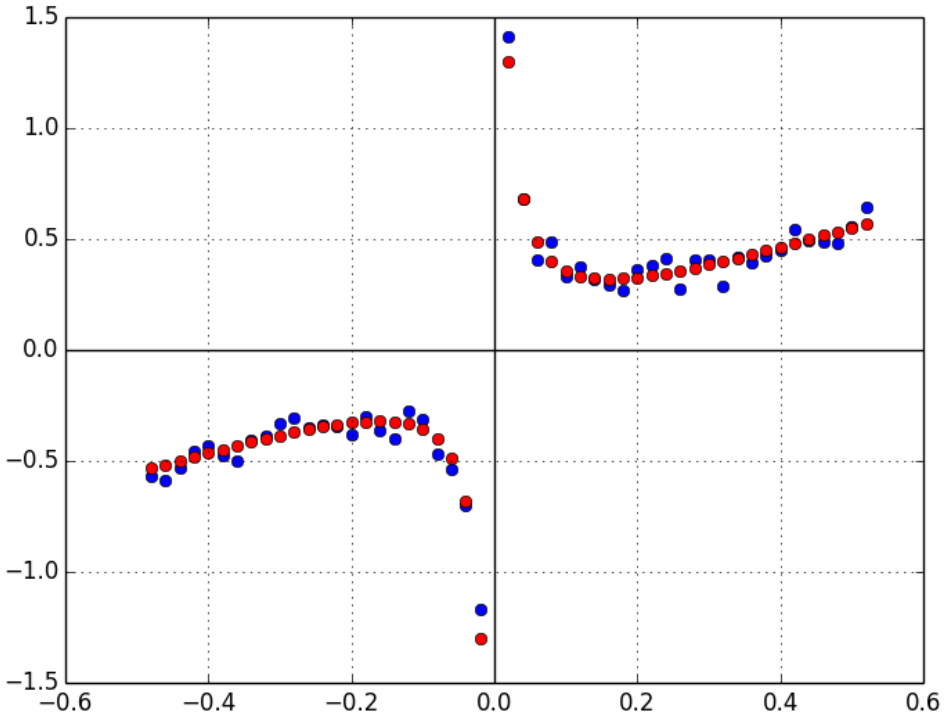
Vermelho: indivíduo

SR_DIV:



Aqui observa-se que os pontos azuis, que representam o dataset, quase não aparecem pois são sobrepostos pelo melhor indivíduo. Este especificamente teve um fitness de 0.02468.

SR_DIV_NOISE:



Aqui observa-se essencialmente que os distúrbios na função não provocaram mudanças drásticas no indivíduo a ser encontrado. A fitness, obviamente, ficou um pouco acima (nesse caso da foto especificamente, uma fitness de 2.16539. Porém a função foi aproximada com sucesso.

Ambos os arquivos estão no diretório de log de execuções. A escolha deles foi aleatória pois para todas as execuções realizadas, os resultados foram parecidos. Nenhuma execução gerou uma fitness maior que 2, para SR_div_noise, e maior que 1, para SR_div.

Após definidas as seeds, ou as configurações para rodar o programa genético sobre o dataset yacht_hydrodynamics, os testes foram realizados.

Para cada configuração, um gráfico foi gerado para descrever a evolução do melhor indivíduo, da média dos indivíduos, do pior, e da porcentagem de filhos melhores que os pais. Como se trata de uma função de 6 variáveis, não é possível desenhá-la no espaço.

Legenda para os gráficos:

Azul: Porcentagem de filhos melhores que os pais ao longo das gerações

Vermelho: os piores indivíduos em cada geração

Amarelo: A média dos indivíduos, por geração

Verde: os melhores indivíduos por geração

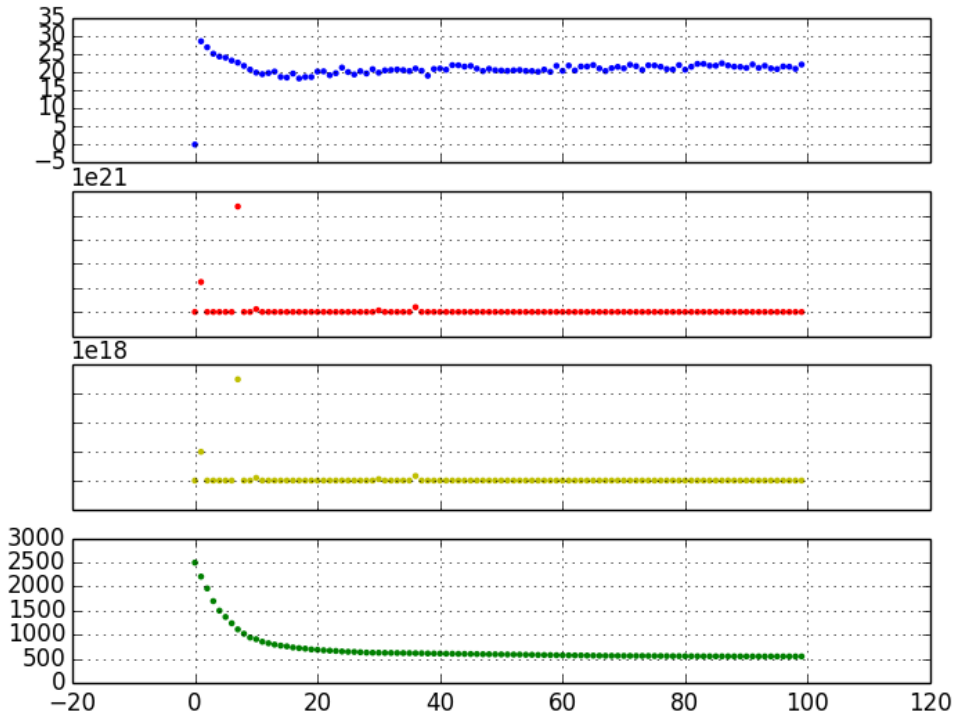
Obs: Os gráficos representam a média dos valores de cada geração em todas as execuções do programa naquela mesma seed. Os gráficos e logs de cada geração também foram gerados, e estão nos diretórios de exec_logs.

Configuração 1:

Probabilidade de Crossover	0.9
Probabilidade de Mutação	0.05
Tamanho do Torneio	20

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
549.085	5.665e+16	1.133e+14	22.14%



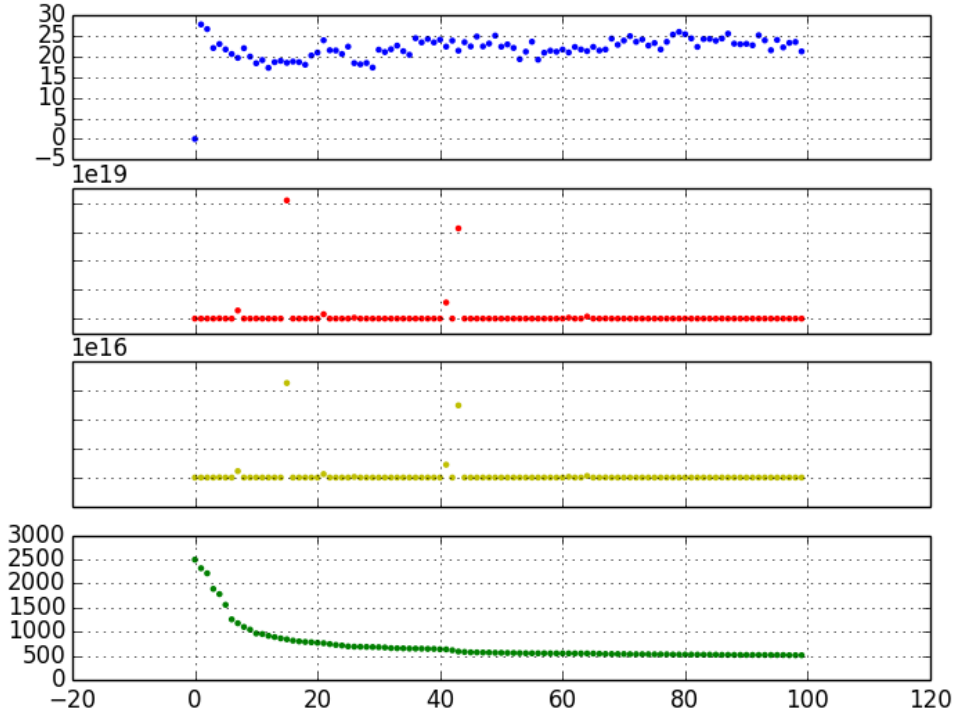
Observa-se que apenas 20% dos filhos se mostrou melhor que os pais, em uma seed na qual aproximadamente 90% dos casais cruzaram. Não é possível observar mudança significativa na média e no pior indivíduo ao longo das gerações devido a discrepância do valor fitness entre eles.

Configuração 2:

Probabilidade de Crossover	0.6
Probabilidade de Mutação	0.3
Tamanho do Torneio	20

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
512.629	6.024e+8	1.499e+6	21.20%



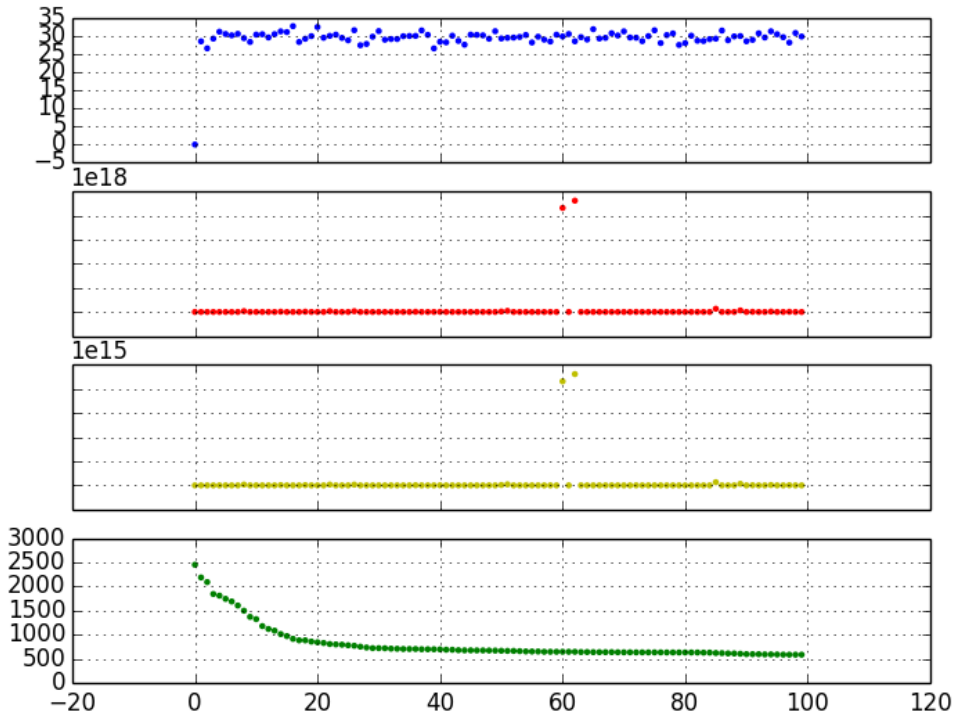
Sem muitas mudanças na porcentagem de crianças que são melhores que os pais, observa-se uma convergência maior em relação a seed anterior. O número de cruzamentos é menor, mas mesmo que a probabilidade de mutação tenha aumentado, a diversidade não aumentou muito, fato sendo evidenciado pela convergência cedo do melhor indivíduo.

Configuração 3:

Probabilidade de Crossover	0.3
Probabilidade de Mutação	0.9
Tamanho do Torneio	20

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
585.799	10.583e+8	2.688e+6	29.90%



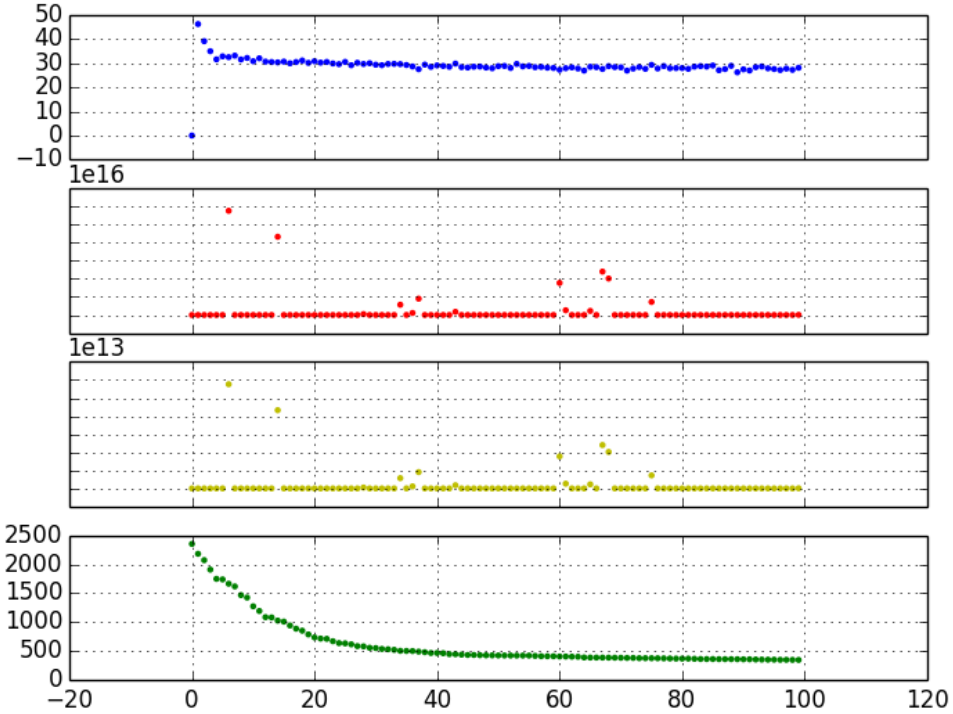
Observa-se que a convergência ocorreu um pouco após a geração 20, e conclui-se que a baixa probabilidade de cruzamentos proporcionou maior diversidade à população. Outro fator importante é que devido a diversidade, e a baixa taxa de cruzamentos, o número de crianças melhores que os pais cresceu em 10% em relação às seeds anteriores. Acredita-se que a alta taxa de mutação tenha propiciado um ambiente em que o cruzamento pouco importou, uma vez que apos o cruzamento, a maioria dos filhos resultantes foi submetido à mutação.

Configuração 4:

Probabilidade de Crossover	0.9
Probabilidade de Mutação	0.05
Tamanho do Torneio	2

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
345.316	3.607e+13	7.214e+10	28.13%



Esta foi a seed que proporcionou melhores resultados para o programa. O torneio de tamanho 2 proporcionou maior diversidade, e retardou a convergência do programa, pois a pressão seletiva foi menor. Foi nessa seed que encontramos o melhor resultado para o problema:

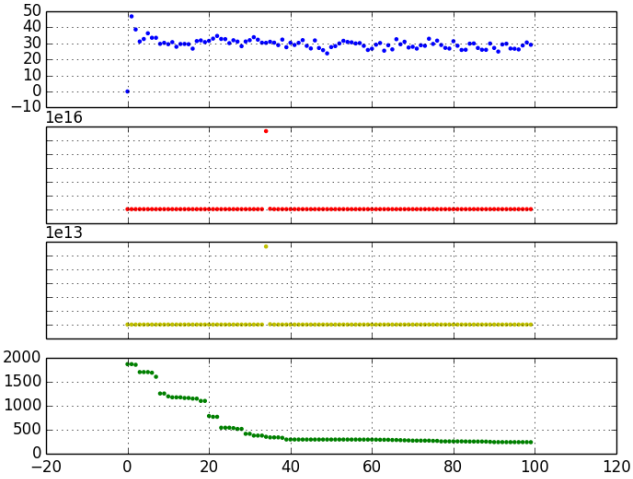
MELHOR INDIVÍDUO ENCONTRADO:

Fitness: 239.515170675

Elapsed time: 7934.36 (s)

Detalhes sobre a função no arquivo:

exec_logs > yacht_hydrodynamics > seed4 > yacht_hydrodynamics.txt_(10-04-2015)_at_19h-34m-44s.txt

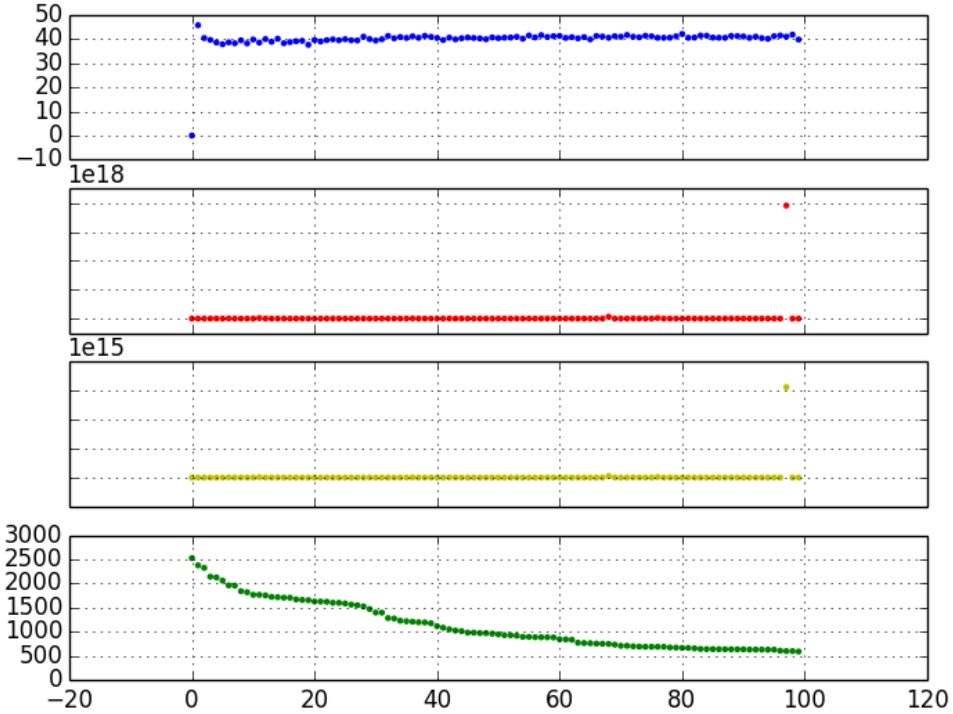


Configuração 5:

Probabilidade de Crossover	0.6
Probabilidade de Mutação	0.3
Tamanho do Torneio	2

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
588.382	8.765e+13	1.753e+11	39.85%



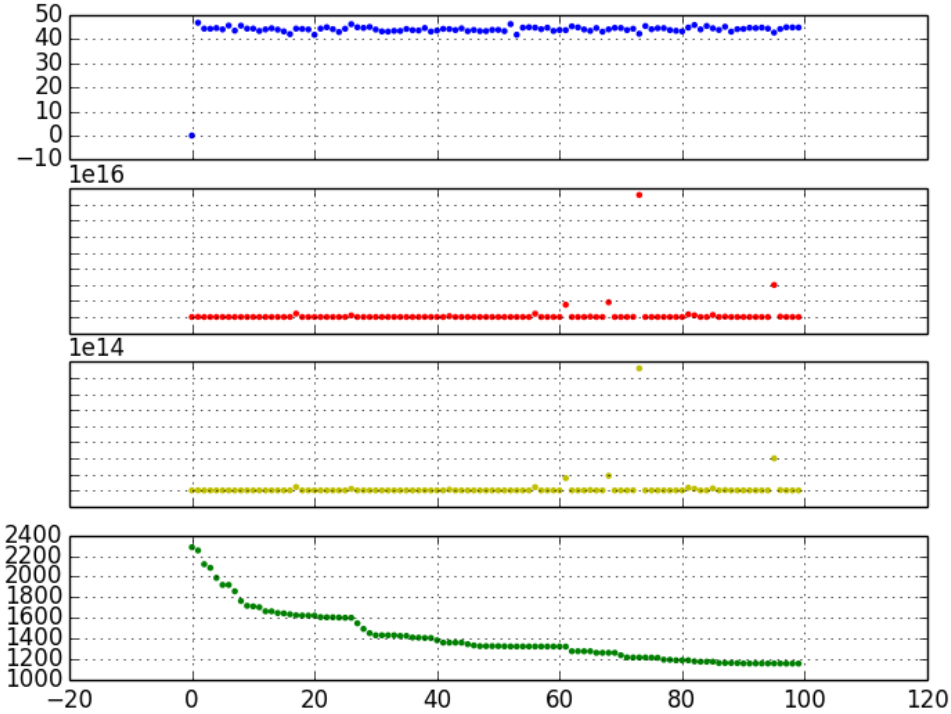
Esta seed não proporcionou bons resultados, porém observa-se que a convergência ocorre apenas após a geração 60. Este bom resultado em relação a convergência se dá devido ao tamanho do torneio (2) e a menor probabilidade de cruzamento (0.6). A presença da mutação ajudou a diversidade, mas não proporcionou indivíduos melhores que a seed anterior. Porém, acredita-se que devido a mutação dos filhos, a porcentagem de filhoes melhores que os pais tenha sofrido esse aumento significativo de 10% em relação a seed anterior, ficando por volta de 40% em cada geração.

Configuração 6:

Probabilidade de Crossover	0.3
Probabilidade de Mutação	0.9
Tamanho do Torneio	2

Resultado médio de todas as gerações 100:

Melhor Ind	Pior Ind	Média Ind	Crianças Melhores (%)
1158.39	1.25e+9	2.82e+6	44.84%



Esta foi a pior seed dentre as 6 selecionadas. Apesar de ter uma baixa convergência, não obteve-se bons resultados de fitness, com todas as execuções terminando com o melhor indivíduo entre 1000 e 1200 de fitness. Acredita-se que a baixa taxa de crossover (mais diversidade) somada à alta taxa de mutação (mais diversidade) e o tamanho pequeno do torneio (menor pressão seletiva e mais diversidade) tenha proporcionado populações caóticas e sem evolução definida, com muita diversidade e pouco aproveitamento de indivíduos bons.

5. CONCLUSÕES

Dado os números absurdos de fitness no caso dos piores indivíduos, e considerando que a média também é alta, conclui-se que o algoritmo sofre de intenso bloating. A presença de introns é enorme, e eles atrasam o algoritmo drasticamente. Uma forma de se medir bloating seria o número de indivíduos extremamente ruins por geração. A presença de indivíduos muito ruins ocorre pela decisão de implementação em complementar o resto da população pelo método ramped half-half, quando um cruzamento não provém uma população inteira.

Observou-se também que o tamanho do torneio foi criteriosamente o parâmetro cujas alterações mais afetaram o programa. Um torneio de tamanho muito grande proporcionou uma pressão seletiva maior, e consequentemente, uma convergência cedo. Já o torneio pequeno (tamanho 2) proporcionou menor pressão seletiva e menor convergência, além de mais diversidade e melhores resultados. A melhor fitness encontrada foi de 239, ao executar o algoritmo sob a configuração número 4.

6. BIBLIOGRAFIA

[1] Banzhaf, W., P. Nordin, R.E. Keller, F.D. Francone. Genetic programming, an introduction, on the automatic evolution of computer programs and its applications. ISBN: 3-920993-58-6, Dpunkt, Heidelberg, Germany, 1998.

[2] http://www2.meteo.uni-bonn.de/staff/venema/essays/2004/genetic_programming_and_bloat.html, Acessado em: 04 de Outubro de 2015.

[3] http://www.genetic-programming.com, Acessado em: 04 de Outubro de 2015.