Otimização Genética Usando Derivados: O Pacote rgenoud para **R**

Walter R. Mebane, Jr.
Universidade de Michigan

Jasjeet S. Sekhon
UC Berkeley

Abstrato

Esta introdução ao pacote R rgenoud é uma versão modificada de Mebane e Sekhon (2011), publicada no Journal of Statistical Software. Essa versão da introdução contém figuras de maior resolução. genoud é uma função R que combina métodos de algoritmos evolucionários com um método baseado em derivadas (quase-Newton) para resolver problemas difíceis de otimização. genoud também pode ser usado para problemas de otimização para os quais não existem derivadas. O genoud resolve problemas não lineares ou talvez até descontínuos nos parâmetros da função a ser otimizada. Quando a função a ser otimizada (por exemplo, uma probabilidade logarítmica) não é linear nos parâmetros do modelo, a função geralmente não será globalmente côncava e pode ter irregularidades como pontos de sela ou descontinuidades. Métodos de otimização que dependem de derivadas da função objetivo podem não encontrar nenhum ótimo. Podem existir vários ótimos locais, de modo que não há garantia de que um método baseado em derivadas irá convergir para o ótimo global. Por outro lado, algoritmos que não usam informações derivadas (como algoritmos genéticos puros) são, para muitos problemas, menos pobres em escaladas locais. A maioria dos problemas estatísticos são regulares em uma vizinhança da solução. Portanto, para alguma parte do espaço de pesquisa, as informações derivadas são úteis. A função oferece suporte ao processamento paralelo em várias CPUs em uma única máquina ou em um cluster de computadores.

Palavras-chave: algoritmo genético, programa evolutivo, otimização, computação paralela, R.

1. Introdução

Desenvolvemos o pacote R rgenoud para resolver problemas de otimização difíceis, como geralmente surgem ao estimar modelos estatísticos não lineares ou resolver funções complicadas não lineares, não suaves e até descontínuas . é uma função não linear dos parâmetros.

Nesses casos, a função a ser otimizada geralmente não é globalmente côncava. Uma função objetivo que não é globalmente côncava pode ter múltiplos ótimos locais, pontos de sela, soluções de contorno ou descontinuidades. Enquanto a função objetivo para um modelo estatístico é muitas vezes côncava em uma vizinhança da solução ótima, essa vizinhança é frequentemente uma pequena proporção do espaço de parâmetros de interesse potencial, e fora dessa vizinhança a função pode ser irregular. Nesses casos, métodos de otimização que dependem inteiramente de derivativos podem

1O pacote de software rgenoud está disponível no Comprehensive R (R Development Core Team 2009) Rede de arquivos em http://CRAN.R-project.org/package=rgenoud.

não são confiáveis e muitas vezes são virtualmente inutilizáveis. Os métodos de Newton-Raphson e quase-Newton estão entre os métodos de otimização comumente usados que dependem completamente de derivadas. Esses métodos funcionam bem quando a função a ser otimizada é regular e suave sobre o domínio dos valores dos parâmetros que são de interesse, mas, caso contrário, os métodos geralmente falham (Gill, Murray e Wright 1981). Mesmo em modelos onde se espera que tais métodos funcionem na maior parte do tempo, técnicas de reamostragem como o bootstrap (Efron e Tibshirani 1994) podem gerar reamostragens nas quais algoritmos de otimização baseados em derivadas encontram sérias dificuldades. Isso é lamentável porque os métodos mais usados para otimização em problemas de estimação estatística são inteiramente baseados em derivativos.

O pacote rgenoud tem sido usado por uma grande variedade de usuários e desenvolvedores. Atualmente, mais de doze pacotes R dependem do rgenoud, incluindo: âncoras (analisando dados de pesquisa com vinhetas de ancoragem, Wand e King 2011; Wand, King e Lau 2008; King and Wand 2007); BARD (redistritamento automatizado, Altman e McDonald 2011); boolean (modelos de resposta binária booleana, Braumoeller 2003; Braumoeller, Goodrich e Kline 2010); DiceOp tim (otimização baseada em kriging para experimentos de computador, Ginsbourger e Roustant 2010); FAiR (análise fatorial, Goodrich 2011); JOP (otimização simultânea de múltiplas respostas, Kuhnt e Rudak 2011); KrigInv (inversão baseada em krigagem, Picheny e Ginsbourger 2011); PKfit (análise de dados em farmacocinética, Lee e Lee 2009a); Matching (escore de propensão e pareamento multivariado, Sekhon 2011b,a); ivivc (modelagem de correlação in vitro-in vivo, Lee e Lee 2009b); multinomRob (modelos multinomiais robustos, Mebane e Sekhon 2009, 2004); e Synth (método de grupo de controle sintético para estudos de caso comparativos, Abadie e Gardeazabal 2003; Diamond e Hainmueller 2008; Abadie, Diamond e Hainmueller 2011).

Apresentamos na Seção 3 um exemplo usando funções de benchmark retiradas de Yao, Liu e Lin (1999), seguido de um exemplo motivado pelo pacote multinomRob. O conjunto de benchmarks inclui funções de alta dimensão, descontínuas ou que possuem muitos ótimos locais.
O pacote multinomRob estima robustamente modelos de regressão multinomial superdispersos e usa rgenoud para resolver um estimador S generalizado de diferença mínima quartil (LQD) (Mebane e Sekhon 2004). O LQD não é uma função suave dos parâmetros do modelo de regressão.
A função é contínua, mas o espaço de parâmetros é finamente particionado por limites não diferenciáveis.

Em outro artigo neste volume, o pacote Matching e seu uso de rgenoud são descritos em detalhes (Sekhon 2011a). A correspondência fornece funções para correspondência multivariada e de pontuação de propensão e para encontrar o equilíbrio covariável ideal com base em um algoritmo de busca genética implementado em rgenoud. A pesquisa sobre correspondências ótimas é descontínua, portanto, não são usadas derivadas.[2] A pesquisa também envolve otimização lexical, que é um recurso exclusivo implementado no rgenoud.[3]

O pacote rgenoud implementa uma versão atualizada e estendida do programa C GENOUD (Mebane e Sekhon 1997) descrito em (Sekhon e Mebane 1998). As muitas melhorias incluem, entre outras coisas, a interface com R, que inclui a capacidade de otimizar funções escritas em R, opções para otimizar parâmetros de ponto flutuante e valores inteiros, o

2A opção BFGS de genoud é definida como FALSE, e o nono operador que depende de derivativos não é usado.

3A otimização lexical é útil quando há vários critérios de adequação; os parâmetros são escolhidos de forma a maximizar os valores de fitness em ordem lexical, ou seja, o segundo critério de ajuste só é considerado se os parâmetros tiverem o mesmo ajuste do primeiro. Veja a opção léxica e Sekhon (2011a) para detalhes. Todas as opções do genoud estão descritas no arquivo de ajuda do R para a função.

capacidade de otimizar funções de perda que retornam vários valores de aptidão (otimização léxica), a capacidade de chamar genoud recursivamente, a capacidade de fazer com que o otimizador avalie apenas para novos valores de parâmetro e a capacidade de usar vários computadores, CPUs ou núcleos para executar tarefas paralelas cálculos.

O programa rgenoud combina um algoritmo evolucionário com um método quase-Newton. O método quase-Newton é o método de Broyden-Fletcher-Goldfarb-Shanno (BFGS) (Gill et al. 1981, 119) implementado na função ótima de R. Quando o BFGS está sendo usado, nosso programa oferece a opção de usar as derivadas numéricas incorporadas do rgenoud (que são baseadas no código retirado de Gill et al. 1981, 337–344) ou derivadas analíticas fornecidas pelo usuário.4 Nosso programa pode também funcionam sem o BFGS, caso em que não são necessárias derivadas e o otimizador funcionará mesmo quando a função for descontínua. O principal benefício do uso de derivadas é que o algoritmo encontrará rapidamente um ótimo local quando um conjunto atual de valores de parâmetros de solução de teste estiver em uma vizinhança suave do ponto ótimo local.
O uso adequado do BFGS pode fazer o algoritmo convergir para o ótimo global muito mais rapidamente. Mas o uso prematuro ou excessivo do BFGS pode impedir a convergência para o ótimo global.5 Como sempre, é perigoso confiar nas configurações padrão de um otimizador. Nosso programa não elimina a necessidade de julgamento, teste e paciência.

Como observam Gill, Murray e Wright, “não há estratégia garantida que resolva todas as dificuldades” (Gill et al. 1981, 285). Neste artigo, revisamos muito brevemente a teoria dos algoritmos de busca aleatória que suporta a afirmação de que rgenoud tem uma alta probabilidade de encontrar o ótimo global quando tal existe. E apresentamos três exemplos de como usar a função genoud: para otimizar um modelo de mistura Normal escalar simples, mas diabólico; minimizar um conjunto de funções de benchmark que foram usadas anteriormente para testar algoritmos de otimização de programação evolutiva; e otimizar uma versão do único estimador LQD intermitentemente diferenciável. Detalhes adicionais sobre a teoria e o desempenho do genoud podem ser encontrados em nosso artigo que descreve o GENOUD (Sekhon e Mebane 1998).

2. Antecedentes em Otimização Genética

Um algoritmo evolucionário (EA) usa uma coleção de regras heurísticas para modificar uma população de soluções de teste de tal forma que cada geração de valores de teste tende a ser, em média, melhor que seu antecessor. A medida para saber se uma solução de teste é melhor que outra é o valor de adequação da solução de teste. Em aplicações estatísticas, a adequação é uma função da estatística de resumo que está sendo otimizada (por exemplo, a probabilidade de log). rgenoud funciona para casos em que uma solução é um vetor de números inteiros ou de ponto flutuante que servem como parâmetros de uma função a ser otimizada. A busca por uma solução ocorre por meio de um conjunto de regras heurísticas, ou operadores, cada um dos quais atua em uma ou mais soluções de teste da população atual para produzir uma ou mais soluções de teste a serem incluídas na nova população. Os EAs não requerem que existam derivadas ou que a função seja contínua para encontrar o ótimo global.

O EA em rgenoud é fundamentalmente um algoritmo genético (GA) no qual as strings de código são vetores de números em vez de strings de bits, e os operadores do GA assumem formas especiais ajustadas para a representação vetorial de ponto flutuante ou inteiro. Um GA usa um conjunto de

4Derivados fornecidos pelo usuário podem ser fornecidos através da opção gr.

5O usuário pode controlar se o genoud usa o BFGS (através da opção BFGS), e se os operadores que usam os BFGS são usados (através da opção P9), com que frequência eles são usados.

operadores genéticos para evoluir uma população finita de sequências de código finitas ao longo de uma série de gerações (Holland 1975; Goldberg 1989; Grefenstette e Baker 1989). Os operadores usados nas implementações de AG variam (Davis 1991; Filho e Alippi 1994), mas em um sentido analítico o conjunto básico de operadores pode ser definido como reprodução, mutação, cruzamento e inversão. A variação desses operadores em diferentes implementações de GA reflete a variedade de códigos mais adequados para diferentes aplicações. A reprodução implica selecionar uma string de código com uma probabilidade que aumenta com o valor de aptidão da string de código. Crossover e inversão usam pares ou conjuntos maiores das strings de código selecionadas para criar novas strings de código. A mutação altera aleatoriamente os valores dos elementos de uma única sequência de código selecionada.

Usados em combinações adequadas, os operadores genéticos tendem a melhorar a aptidão média de cada geração sucessiva, embora não haja garantia de que a aptidão média melhorará entre cada par de gerações sucessivas. A aptidão média pode muito bem diminuir. Mas existem teoremas para provar que partes das soluções experimentais que têm valores de aptidão acima da média na população atual são amostradas a uma taxa exponencial para inclusão na população subsequente (Holland 1975, 139-140). A população de cada geração contém uma amostra tendenciosa de strings de código, de modo que o desempenho de uma substring nessa população é uma estimativa tendenciosa de seu desempenho médio em todas as populações possíveis (De Jong 1993; Grefenstette 1993).

As propriedades de longo prazo de um AG podem ser entendidas pensando no AG como uma cadeia de Markov. Um estado da cadeia é uma população de strings de código do tamanho usado no GA. Para cadeias de código de comprimento finito e populações GA de tamanho finito, o espaço de estados é finito. Se tal AG usa reprodução aleatória e mutação aleatória, todos os estados sempre têm uma probabilidade positiva de ocorrer. Um AG finito com reprodução e mutação aleatória é, portanto, aproximadamente uma cadeia de Markov finita e irredutível.6 Uma cadeia de Markov finita e irredutível converge a uma taxa exponencial para uma distribuição estacionária única (Billingsley 1986, 128). Isso significa que a probabilidade de que cada população ocorra rapidamente converge para um valor constante e positivo.
Nix e Vose (1992) e Vose (1993) usam um modelo de cadeia de Markov para mostrar que em um AG onde a probabilidade de cada string de código ser selecionada para reproduzir é proporcional à sua aptidão observada, a distribuição estacionária enfatiza fortemente as populações que contêm código -strings que possuem valores de fitness altos. Eles mostram que assintóticas no tamanho da população – ou seja, no limite para uma série de AGs com populações sucessivamente maiores – populações que têm aptidão média subótima têm probabilidades próximas de zero na distribuição estacionária, enquanto a probabilidade para a população que tem aptidão média ótima se aproxima de um. Se k > 1 populações têm aptidão média ótima, então na distribuição estacionária limitante a probabilidade para cada uma se aproxima de 1/k.

Os resultados teóricos de Nix e Vose implicam que o sucesso de um GA como otimizador depende de ter uma população suficientemente grande de strings de código. Se a população do AG não for suficientemente grande, então a cadeia de Markov que o AG implementa aproximadamente está convergindo para uma distribuição estacionária na qual as probabilidades de estados ótimos e subótimos não são claramente distinguidas. Populações subótimas podem ser tão prováveis de ocorrer quanto as ótimas. Se a distribuição estacionária não for favorável, o tempo de execução em termos de gerações necessário para produzir uma sequência de código ótima será excessivo. Para todos os espaços de estados, exceto trivialmente pequenos, um

6Feller (1970, 372-419) e Billingsley (1986, 107-142) revisam as propriedades relevantes das cadeias de Markov.
A aleatoriedade em um AG real depende do desempenho de geradores de números pseudoaleatórios. Isso e as limitações da aritmética de ponto flutuante significam que não é literalmente verdade que um AG real tenha uma probabilidade positiva de atingir qualquer estado a partir de qualquer outro estado, e alguns estados podem de fato não ser alcançáveis a partir de um determinado conjunto de condições iniciais.

distribuição estacionária desfavorável pode facilmente implicar em um tempo de execução esperado em milhões de gerações. Mas se a distribuição estacionária enfatiza fortemente as populações ótimas, relativamente poucas gerações podem ser necessárias para encontrar uma sequência de código ótima. Em geral, a probabilidade de produzir um ótimo em um número fixo de gerações tende a aumentar com o tamanho da população do AG.

O algoritmo evolucionário no rgenoud usa nove operadores listados na Tabela 1. Os operadores estendem e modificam um conjunto de operadores usados no GENOCOP (Michalewicz, Swaminathan e Logan 1993; Michalewicz 1992). Os operadores são numerados usando a correspondência de sintaxe que costumava se referir a eles por rgenoud. O operador de clonagem simplesmente faz cópias da melhor solução de teste na geração atual (independente desse operador, rgenoud sempre retém uma instância da melhor solução de teste). Os operadores de mutação uniforme, mutação de limite e mutação não uniforme atuam em uma única solução de teste. A mutação uniforme altera um parâmetro na solução de teste para um valor aleatório uniformemente distribuído no domínio especificado para o parâmetro. A mutação de limite substitui um parâmetro por um dos limites de seu domínio. A mutação não uniforme reduz um parâmetro em direção a um dos limites, com a quantidade de redução diminuindo à medida que a contagem de geração se aproxima do número máximo especificado de gerações. A mutação não uniforme completa faz a mutação não uniforme para todos os parâmetros na solução de teste. O cruzamento heurístico usa duas soluções de teste para produzir uma nova solução de teste localizada ao longo de um vetor que começa em uma solução de teste e aponta para longe da outra. O cruzamento de politopos (inspirado na pesquisa simplex, Gill et al. 1981, 94-95) calcula uma solução de teste que é uma combinação convexa de tantas soluções de teste quantos parâmetros existem. O cruzamento simples calcula duas soluções de teste a partir de duas soluções de teste de entrada trocando valores de parâmetro entre as soluções depois de dividir as soluções em um ponto selecionado aleatoriamente. Este operador pode ser especialmente eficaz se a ordenação dos parâmetros em cada solução de teste for conseqüente. O cruzamento de mínimo local calcula uma nova solução de teste em duas etapas: primeiro, ele faz um número predefinido de iterações de BFGS a partir da solução de teste de entrada; em seguida, ele calcula uma combinação convexa das soluções de entrada e a iteração do BFGS.

3. Exemplos

A única função no pacote rgenoud é genoud. A interface desta função é semelhante à da função ótima em R. Mas a função tem muitos argumentos adicionais que controlam o comportamento do algoritmo evolucionário.

3.1. Garra Dupla Assimétrica:

Nosso primeiro exemplo, que também estudamos em Sekhon e Mebane (1998), é uma mistura normal chamada Asymmetric Double Claw (ADC). Traçamos a função na Figura 1. Matematicamente, esta mistura é definida como

$$fADC = P1 \sum_{m=0}^{46} \frac{1}{100N}\, 2^{m-1},\ \frac{2}{3} + P3 \sum_{m=1}^{} \frac{1}{300N}\, \frac{m}{2},\ \frac{1}{100} + P3 \sum_{m=1}^{} \frac{7}{300N}\, \frac{m}{2},\ \frac{7}{100}, \qquad (1)$$

onde N é a densidade normal.

A garra dupla assimétrica é difícil de maximizar porque existem muitas soluções locais.
Há cinco máximos locais na Figura 1. Os otimizadores padrão baseados em derivativos simplesmente subiriam a colina mais próxima do valor inicial.

Para otimizar essa mistura normal devemos primeiro criar uma função para ela

Clonagem P1. Copie Xt para a próxima geração, Xt+1.

Mutação Uniforme P2. Escolha aleatoriamente i ÿ N. Selecione um valor ˜xi ÿ U(xi , xi). Defina Xi = ˜xi .

Mutação de limite P3. Escolha aleatoriamente i ÿ N. Defina Xi = xi ou Xi = xi probabilidade 1/2 de usar , com cada valor.

P4 Mutação não uniforme. Escolha aleatoriamente i ÿ N. Calcule p = (1 ÿ t/T) Bu, onde t é o número de geração atual, T é o número máximo de gerações, B > 0 é um parâmetro de ajuste e u ÿ U(0, 1). Defina Xi = (1ÿp)xi+pxi ou Xi = (1ÿp)xi+pxi , com probabilidade 1/2 de usar cada valor.

P5 Polytopo Crossover. Usando m = max(2, n) vetores x da população atual e m números aleatórios pj ÿ (0, 1) tais que Pm j=1 pj = 1, conjunto X = Pm j=1 pjxj .

P6 Cruzamento Simples. Escolha um único inteiro i de N. Usando dois vetores de parâmetro, xey, defina Xi = pxi + (1 ÿ p)yi e Yi = pyi + (1 ÿ p)xi , onde p ÿ (0, 1) é um Número fixo.

P7 Mutação Não Uniforme Inteira. Faça mutação não uniforme para todos os elementos de X.

Cruzamento heurístico P8. Escolha p ÿ U(0, 1). Usando dois vetores de parâmetros, xey, calcule z = p(xÿy)+x. Se z satisfizer todas as restrições, use-o. Caso contrário, escolha outro valor p e repita. Defina z igual ao melhor de xey se um z misto satisfatório não for encontrado por um número predefinido de tentativas. Desta forma, produza dois vetores z.

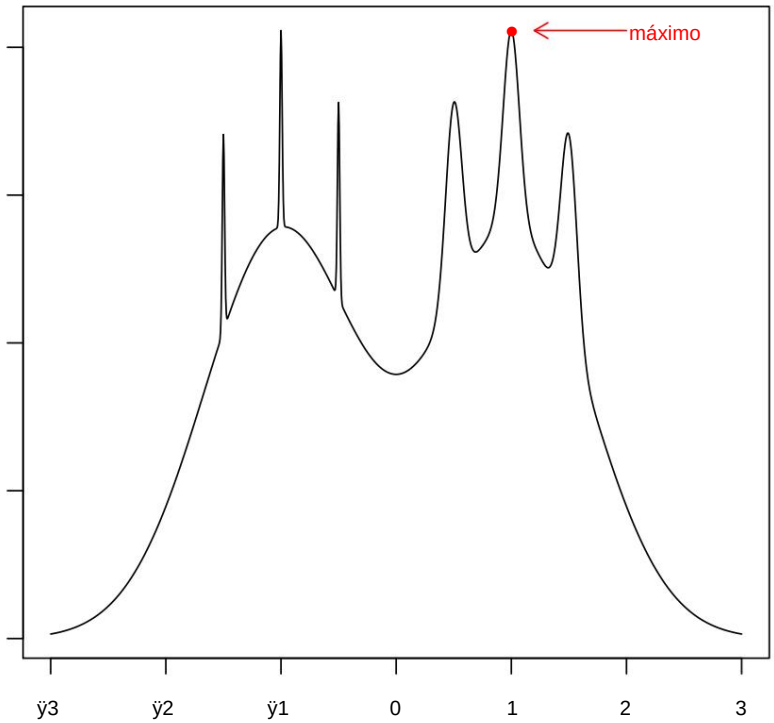
P9 Crossover mínimo local. Escolha p ÿ U(0, 1). Começando com x, execute a otimização BFGS até um número predefinido de iterações para produzir ˜x. Calcule z = p˜x + (1 ÿ p)x.
Se z satisfizer as restrições de limite, use-o. Caso contrário, reduza p definindo p = p/2 e recalcule z. Se um z satisfatório não for encontrado por um número predefinido de tentativas, retorne x.
Este operador é extremamente computacionalmente intensivo, use com moderação.

Notação:

X = X1, . . . , Xn é o vetor de n parâmetros Xi . xi é o limite inferior e xi é o limite superior nos valores de Xi . xi é o valor atual de Xi e x é o valor atual de X.
N = {1, . . . , n}. p ÿ U(0, 1) significa que p é extraído da distribuição uniforme no intervalo [0, 1].

Tabela 1: Operadores 'genoud'. Adaptado de Sekhon e Mebane (1998).

Figura 1: Mistura Normal: Garra Dupla Assimétrica




```
>         garra <- função(xx) {
+           x <- xx[1]
+             y <- (0,46*(dnorm(x,-1,0,2,0/3,0) + dnorm(x,1,0,2,0/3,0)) +
+              (1.0/300.0)*(dnorm(x,-0.5,.01) + dnorm(x,-1.0,.01) + dnorm(x,-1.5,.01)) +
+             (7.0/300.0)*(dnorm(x,0.5,.07) + dnorm(x,1.0,.07) + dnorm(x,1.5,.07)))
+              retorno(s)
+         }
```

E agora fazemos uma chamada para rgenoud usando esta função:

```
> biblioteca("rgenoud")
> garra1 <- genoud(garra, nvars=1, max=TRUE, pop.size=3000)
```

O primeiro argumento de genoud é a função a ser otimizada. O primeiro argumento desse
A função deve ser o vetor de parâmetros sobre os quais a otimização deve ocorrer. Geralmente, o
a função deve retornar um resultado escalar.7 O segundo argumento de genoud neste exemplo
(nvars) é o número de variáveis que a função a ser otimizada leva. O terceiro argumento,
max=TRUE, diz ao genoud para maximizar a função em vez de seu comportamento padrão que é
minimizar.

A quarta opção pop.size controla a parte mais importante do algoritmo evolutivo,
o tamanho da população. Este é o número de indivíduos que o genoud usa para resolver a otimização

7A função a ser otimizada pode retornar um vetor caso se deseje fazer otimização lexical. por favor veja o opção lexical para genoud.

problema. Como observado na discussão teórica, os teoremas relacionados aos algoritmos evolutivos são assintóticos no tamanho da população, então quanto maior é geralmente melhor, mas obviamente leva mais tempo. A solução máxima da densidade de garra dupla é encontrada de forma confiável pelo genoud mesmo usando o valor padrão de pop.size=1000. A confiabilidade aumenta à medida que o pop.size aumenta. Infelizmente, devido à natureza estocástica do algoritmo, é impossível responder geralmente à questão de qual é o melhor tamanho de população a ser usado.

Outras opções determinam o número máximo de gerações que o algoritmo evolucionário calcula. Essas opções são max.generations, wait.generations e hard.generation.limit.
O ponto de terminação especificado também afeta o desempenho de alguns dos operadores: os dois operadores de mutação não uniformes introduzem intervalos menores de variação nas soluções de teste à medida que a contagem de geração se aproxima do valor especificado max.generations. Existem muitas outras opções que podem ser usadas para ajustar o comportamento do algoritmo.

A saída impressa por genoud é controlada por um argumento print.level. O valor padrão, print.level=2, produz uma saída relativamente detalhada que fornece informações extensas sobre o conjunto de operadores que está sendo usado e o andamento da otimização. Normalmente, as convenções do R sugerem definir o padrão como print.level=0, o que suprimiria a saída para a tela, mas como as execuções do genoud podem levar muito tempo, pode ser importante que o usuário receba algum feedback para ver que o programa não morreu e ser capaz de ver onde o programa parou se eventualmente não conseguir progredir adequadamente.

A saída impressa pela chamada anterior de genoud, que usa o valor padrão para um argumento print.level, é a seguinte.

```
Sex 9 de fevereiro 19:33:42 2007
Domínios:
  -1,000000e+01 <= X1 <=                    1.000000e+01

Tipo de dados: Operadores de ponto
flutuante (número de código, nome, população)
                (1) Clonagem ..................... 372 (2) Mutação Uniforme ..... .....
                375 (3) Mutação de Contorno ..... 375 (4) Mutação Não
                Uniforme ..... 375 (5) Cruzamento Politopo..... 375 (6)
                Cruzamento Simples..... 376 (7) Mutação Não Uniforme
                Inteira..... 375 (8) Cruzamento Heurístico..... 376 (9)
                Cruzamento Mínimo Local..... ... 0

HARD Número Máximo de Gerações: 100
Máximo de Gerações Inalteráveis: 10 Tamanho da
população: 3000 Tolerância de Convergência: 1.000000e-03

Usando o otimizador baseado em derivativos BFGS no melhor indivíduo de cada geração.
Verificando gradientes antes de parar.
Usando Indivíduos Fora dos Limites.
```

Problema de maximização.

GERAÇÃO: 0 (inicializando a população)
Valor de aptidão... 4.112017e-01
média..... 4.990165e-02
variação..... 9.708147e-03
#unique..... 3000, #Total UniqueCount: 3000 var 1:

melhor..... 9.966758e-01
média..... 3.453097e-02
variação ..... 3,373681e+01

GERAÇÃO: 1
Valor de aptidão... 4.113123e-01
média..... 2.237095e-01
variação..... 2.566140e-02
#unique..... 1858, #Total UniqueCount: 4858 var 1:

melhor..... 9.995043e-01
média..... 4.615946e-01
variação ..... 7,447887e+00

[...]

GERAÇÃO: 10
Valor de aptidão... 4.113123e-01
média..... 2.953888e-01
variação..... 2.590842e-02
#unique..... 1831, #Total UniqueCount: 21708
var 1:
melhor..... 9.995033e-01
média..... 8.403935e-01
variação ..... 5,363241e+00

GERAÇÃO: 11
Valor de aptidão... 4.113123e-01
média..... 2.908561e-01
variação..... 2,733896e-02
#unique..... 1835, #Total UniqueCount: 23543 var 1:

melhor..... 9.995033e-01
média..... 8.084638e-01
variação..... 6,007372e+00

limite de 'wait.generations' atingido.
Nenhuma melhoria significativa em 10 gerações.

Valor de adequação da solução: 4.113123e-01

Parâmetros na Solução (parâmetro, gradiente):

X[1]: 9.995033e-01 G[1]: -6.343841e-09

Solução encontrada geração 1

Número de Gerações Executadas 11

Sex 9 de fevereiro 19:33:45 2007

Tempo total de execução: 0 horas 0 minutos e 3 segundos

Após imprimir a data e hora da execução, o programa imprime o domínio de valores que está permitindo para cada parâmetro da função que está sendo otimizada. Neste caso, os valores de domínio padrão estão sendo usados. Naturalmente, é importante especificar domínios amplos o suficiente para incluir a solução. Na prática, com funções altamente não lineares, muitas vezes é melhor especificar domínios que são relativamente amplos do que ter domínios que limitam estreitamente e talvez até corretamente a solução. Esse comportamento surpreendente reflete o fato de que com uma função altamente não linear, um ponto que está próximo da solução no sentido de simples proximidade numérica pode não estar tão próximo no sentido de haver um caminho curto viável para se chegar à solução.

Em seguida, o programa imprime o Tipo de Dados. Isso indica se os parâmetros da função a ser otimizada estão sendo tratados como números de ponto flutuante ou inteiros. Para obter mais informações sobre isso, consulte o argumento data.type.int.

O programa então exibe o número de operadores que estão sendo usados, seguidos pelos valores que descrevem as outras características definidas para esta corrida em particular: o número máximo de gerações, o tamanho da população e o valor de tolerância a ser usado para determinar quando os valores dos parâmetros serão ser considerado convergente.

A saída então informa se a otimização do BFGS será aplicada à melhor solução de teste produzida pelas operadoras em cada geração. Para problemas que são suaves e côncavos em uma vizinhança do ótimo global, usar o BFGS dessa maneira pode ajudar o genoud a convergir rapidamente, uma vez que a melhor solução de teste está na vizinhança correta. Esta execução de genoud também calculará o gradiente na melhor solução de teste antes de parar. Na verdade, essa verificação de gradiente é usada como uma verificação de convergência. O algoritmo não começará a contar seu conjunto final de gerações (o wait.generations) até que cada elemento do gradiente seja menor em magnitude do que a tolerância de convergência. Os gradientes nunca são usados e a otimização BFGS não é usada quando os parâmetros da função a ser otimizada são inteiros.

A próxima mensagem descreve quão estritamente genoud está aplicando as restrições de limite especificadas pelos valores de domínio. Por padrão (boundary.enforcement=0), as soluções de teste podem vagar livremente fora dos limites. Os limites são usados apenas para definir domínios para os operadores que usam as informações de limite. Outras configurações do argumento border.enforcement induzem uma aplicação mais rigorosa ou completamente estrita das restrições de limite. Observe que as restrições de limite se aplicam aos parâmetros um de cada vez. Para impor restrições que são definidas por relacionamentos funcionais ou dependentes de dados mais complicados, pode-se incluir uma função de penalidade apropriada como parte da definição da função a ser otimizada, permitindo que essa função defina um valor extremamente alto (se minimizado) ou baixo (se maximizando) valor a ser retornado se as condições desejadas forem violadas.

Depois de relatar se está resolvendo um problema de minimização ou maximização, o genoud relata estatísticas resumidas que descrevem a distribuição do valor de aptidão e os valores dos parâmetros nas soluções de teste no final de cada geração. No caso padrão em que o genoud está acompanhando todas as soluções distintas consideradas ao longo de toda a execução de otimização, esses relatórios de geração também incluem um relatório do número de soluções de teste únicas na população atual e o número de soluções únicas já consideradas . A vantagem de acompanhar as soluções é evitar avaliar repetidamente a função que está sendo otimizada para o conjunto idêntico de valores de parâmetro. Isso pode ser uma eficiência importante quando as avaliações de função são caras, como podem ser em aplicações estatísticas onde os dados são extensos. Esse comportamento de rastreamento é controlado pelo argumento MemoryMatrix.

Na convergência, ou quando o limite máximo de geração é atingido, o programa imprime o valor de aptidão na melhor solução de teste e os valores dos parâmetros dessa solução. Neste caso, a solução foi encontrada após uma geração. Embora o Asymmetric Double Claw possa apresentar um desafio difícil para um otimizador baseado em gradiente que usa apenas subidas locais, é um problema quase trivialmente simples para o genoud.

3.2. Funções de referência:

O segundo exemplo é um conjunto de 23 funções não lineares de referência usadas em Yao et al. (1999) para estudar um par de algoritmos de otimização de programação evolutiva. As definições de função estão na Tabela 2. Como ela inclui um componente aleatório e, portanto, não possui um valor mínimo reproduzível, ignoramos a função numerada 7 em sua sequência.8 As implementações dessas funções estão disponíveis no arquivo R suplementar fornecido com este artigo.9 Essas definições de R incluem os valores das constantes usadas nas funções 14, 15 e 19 a 23. Os domínios dos argumentos da função são restritos aos domínios específicos usados por Yao et al. (1999) por meio de limites declarados na lista denominada testbounds no arquivo suplementar.

Como Yao et ai. (1999) descrevem, otimizar cada uma das funções 1-13 apresenta um problema de alta dimensão. Cada uma dessas funções tem n = 30 parâmetros livres. As funções de 1 a 5 são unimodais, sendo a função 5 uma versão de 30 dimensões da função de Rosenbrock em forma de banana.
A função 6 é uma função degrau. A função 6 tem um valor mínimo que ocorre quando todos os argumentos estão no intervalo xi ÿ [0, .5), e a função é descontínua. As funções 8–13 são multimodais, definidas de tal forma que o número de mínimos locais aumenta exponencialmente com o número de argumentos. Yao et ai. (1999, 84) descrevem essas funções como entre “a classe mais difícil de problemas para muitos algoritmos de otimização (incluindo [programação evolutiva])”. As funções 14–23, que têm entre dois e seis parâmetros livres cada, têm apenas alguns mínimos locais. No entanto, os algoritmos de programação evolutiva considerados por Yao et al. (1999) têm problemas para otimizar as funções 21–23. Embora Yao et al. (1999, 85) afirmam que cada uma dessas funções tem um valor mínimo de -10, mais de 50 replicações os dois algoritmos que eles consideram alcançar soluções com média entre -5,52 e -9,10 (Yao

8A função omitida é $\sum_{i=1}^{n} ix^4 + U(0, 1)$, onde U(0, 1) é uma variável aleatória uniformemente distribuída no intervalo unitário que assume um novo valor sempre que a função é avaliada. Este aspecto estocástico significa que mesmo dado o conjunto de parâmetros que minimizam o componente não estocástico $\sum_{i=1}^{n} ix^4$, ou seja, xi = 0, o valor da função virtualmente nunca atinge o valor mínimo possível de zero. Um otimizador que avaliasse a função em xi = 0 não obteria, em geral, um valor de função menor que o valor de função obtido para uma ampla faixa de valores de parâmetros diferentes. Portanto, não consideramos essa função como um bom teste para algoritmos de otimização de função.

9Este arquivo suplementar está disponível em http://sekhon.berkeley.edu/rgenoud/func23.R.

fun. definição 1 Pn		n mínimo 30 0
i=1 x eu 2		
2 Pn 3 Pn i=1 |xi | + Qn i=1 |xi |		30 0
i=1(Pi j=1 xj) maxi{|xi |,		30 0
4 1 ÿ i ÿ n}		30 0
5 Pnÿ1 i=1 [100(xi+1 ÿ x 6 2 eu)2 + (xi ÿ 1)2],		30 0
Pn i=1(bxi + 0,5c)		30 0
7 Pn i=1 ix4 eu + U(0, 1)		30 0
8 Pn i=1 ÿxi sen(p |xi |)		30 ÿ12569,5
9 Pn i=1[x 2 eu ÿ 10 cos(2ÿxi) + 10]		30 0
10 ÿ20 exp ÿ0,2 q nÿ1 Pn i = 1 i 2 x ÿ exp(n ÿ1 Pn i=1 cos 2ÿxi) + 20 + e		30 0
11 (1/1000)Pn i=1 x eu 2 ÿ Qn i=1 cos xi/ ÿ i + 1		30 0
12 n ÿ1ÿ n 10 sen2 (ÿy1) + Pnÿ1 i=1 (yi ÿ 1)2 [1 + 10 sen2 (ÿyi+1)] + (yn ÿ 1)2 o + Pn i=1 u(xi , 10, 100, 4), yi = 1 + (xi + 1)/4, u(xi , a, k, m) = ÿ k(xi ÿ a) m, xi > a, 0, ÿa ÿ xi ÿ a, yy ÿÿ k(ÿxi ÿ a) m, xi < ÿa,		30 0
13 n sin2 (3ÿx1) + Pnÿ1 i=1 (xi ÿ 1)2 [1 + sin2 (3ÿxi+1)]o /10 +(xn ÿ 1)[1 + sin2 (2ÿxn)]/10 + Pn i=1 u(xi , 5, 100, 4)		30 0
14 n 1/500 + P25 j=1 1/ hj + P2 i=1(xi ÿ aij) 6 ioÿ1		2 1
15 P11 i=1[ai ÿ x1(b 2 + bix2)/(b 2 eu + bix3 + x4)]2		4 0,0003075
16 2 4x 1 4 ÿ 2,1 x 1 2 6 + x 1 2 e /3 + x1x2 ÿ 4x 2 2 4 + 4 x 2		2 ÿ1,0316285
17 [x2 ÿ 5,1x 1/(4ÿ) + 5x1/ÿ ÿ 6]2 + 10[1 ÿ 1/(8ÿ)] cos(x1) + 10		2 0,398
18 [1 + (x1 + x2 + 1)2 (19 ÿ 14x1 + 3x 2 1 ÿ 14x2 + 6x1x2 + 3x 2 2 ×[30 + (2x1 ÿ 3x2) 2 (18 ÿ 32x1 + 12x 2 1)] + 48x2 ÿ 36x1x2 + 27x 2 2)]		2 3
19 ÿ P4 i=1 ci exp h ÿ P4 i=1 ci j=1 aij (xi ÿ pij) 2 eu		4 ÿ3,86
20 ÿ P4 exp h ÿ P6 j=1 aij (xi ÿ pij) 2 eu		6 ÿ3,32
21 P5 i=1[(x ÿ ai) 0 (x ÿ ai) + ci] ÿ1		4-10
22 P7 i=1[(x ÿ ai) 0 (x ÿ ai) + ci] ÿ1		4-10
23 P10 i=1[(x ÿ ai) 0 (x ÿ ai) + ci] ÿ1		4-10

Notas: aValor mínimo da função dentro dos limites especificados, conforme fornecido por Yao et al. (1999, 85).

Tabela 2: 23 Funções de Referência

et ai. 1999, 88, Tabela IV).

Usamos essas funções de referência para ilustrar não apenas como o genoud pode ser eficaz com uma série de problemas difíceis, mas também para enfatizar um aspecto importante de como se deve pensar em tentar ajustar o desempenho do otimizador. A teoria sobre algoritmos genéticos sugere que as soluções ótimas são mais prováveis de aparecer à medida que o tamanho da população de soluções candidatas e o número de gerações aumentam. Em genoud, dois argumentos determinam o número de gerações. Um é max.generations: se hard.generation.limit=TRUE, então o valor especificado para max.generations é um limite superior de ligação. A outra é wait.generations, que determina quando o algoritmo termina se hard.generation.limit=FALSE. Mas mesmo se hard.generation.limit=TRUE, então wait.generations determina por quantas gerações o algoritmo continua uma vez que o melhor vetor de parâmetros e o valor da função que está sendo otimizada parecem ter se estabilizado. O fato de que a melhor solução atual não está mudando não deve ser tratado como decisivo, porque esta solução pode ser apenas um ótimo local ou um ponto de sela. Se o tamanho da população for suficiente, o algoritmo tende a construir uma população de soluções de teste que contém parâmetros nas vizinhanças de todos os ótimos locais competitivos no domínio definido pelos limites dos parâmetros. Mesmo enquanto a melhor solução atual é estável, o algoritmo está melhorando as soluções próximas a outros ótimos locais. Portanto, ter um valor de espera.gerações mais alto nunca piora a eficácia do algoritmo.

Aumentar max.generations pode ou não melhorar a otimização. O valor de max.generations define o valor de T usado nos operadores de mutação - operadores 4 e 7 na Tabela 1. Esses operadores de mutação realizam busca aleatória perto de uma solução de teste que foi selecionada para mutação somente quando a contagem de geração atual é apreciável fração de T. Portanto, aumentar max.generations sem alterar wait.generations aumenta o período durante o qual a busca aleatória está ocorrendo em um domínio mais amplo. Para funções multimodais, uma pesquisa mais ampla pode ser útil, mas às vezes não é bom fazer uma pesquisa mais densa perto das soluções de teste atuais.

Usamos genoud para minimizar as 23 funções usando dois valores para pop.size (5000 e 10000) e dois valores para max.generations (30 e 100). Seguindo Yao et al. (1999), replicamos cada otimização 50 vezes. O código a seguir descreve os cálculos. A lista testfuncs, vetor testNparms e list testbounds são definidos no arquivo R suplementar, e supõe-se que este arquivo seja carregado com o comando source("supplemental.R"). O vetor gradcheck é verdadeiro para todos os elementos, exceto os correspondentes às funções 6 e 7.

```
> source("supplemental.R") >
gradcheck <- rep(TRUE,23) >
gradcheck[6:7] <- FALSE > sizeset
<- c(5000,10000) > genset <-
c(30,100) > nreps < - 50 > gsarray
<- array(NA, dim=c(length(sizeset),
length(genset), 23, nreps)) > asc <- function(x) { as.character(x) } > dimnames(gsarray) <-
lista(asc(conjunto de tamanhos), asc(conjunto de geradores), NULL, NULL); > for (gsize in
sizeset) { + for (ngens in genset) { for (i in 1:23) { for (j in 1:nreps) {


+
+
```

```
+                 gsarray[as.character(gsize), as.character(ngens),i,j] <-
+                     genoud(testfuncs[[i]], nvars=testNparms[i], pop.size=gsize, max.gen=ngens,
+                             hard.gen=TRUE, Domains=testbounds[[i]], solution.tol=1e-6 , limite=1,
+                             gradient.check=gradcheck[i], print=0)$valor
+
+             }
+         }
+ } + }
```

Usar genoud para minimizar as funções de benchmark produz excelentes resultados, pelo menos quando os argumentos pop.size e max.generations são suficientemente grandes. A Tabela 3 relata os valores médios das funções para cada configuração dos argumentos. Esses valores podem ser comparados tanto com os mínimos da função verdadeira dados por Yao et al. (1999) (veja a coluna mais à direita na Tabela 2) e aos valores médios mínimos da função Yao et al. (1999) relatam para seu algoritmo de programação evolucionária “rápida” (FEP), que aparecem na última coluna da Tabela 3. As médias de genoud para as configurações max.gen=100 são iguais ou próximas aos mínimos verdadeiros para todas as funções, exceto função 13. Pode-se argumentar razoavelmente que as soluções médias para a função 5 não são tão próximas de zero quanto seria desejável: essas médias estão próximas de 10ÿ7 enquanto as médias para outras funções que têm um mínimo verdadeiro de zero são 10ÿ15 ou menor. E as médias para as funções 6, 12 e 15 no caso pop.size=5000 estão desativadas.

O efeito de aumentar pop.size é aparente em relação a essas três funções e também às funções 13 e 20–23: os mínimos médios são menores com pop.size=10000 do que com pop.size=5000. Exceto para as funções 6 e 12 no caso pop.size=5000 e função 13, todas as médias de solução genoud para max.gen=100 são ligeiramente ou substancialmente melhores do que as médias de solução FEP correspondentes.

Os resultados na Tabela 3 ilustram claramente as consequências potenciais de não permitir que o genoud seja executado por um número suficiente de gerações. Embora algumas das soluções genoud para max.gen=30 tenham médias competitivas, várias das médias não são nada boas.

O efeito do aumento do pop.size é ainda mais evidente na Tabela 4, que relata os desvios padrão dos respectivos mínimos nas 50 repetições. Com exceção das funções 6, 12, 13, 15 e 21 com pop.size=5000, as soluções genoud para max.gen=100 variam muito menos do que as soluções FEP correspondentes. Para essas funções e também para as funções 20, 22 e 23, as soluções max.gen=100 com pop.size=10000 variam notavelmente menos do que as soluções com pop.size=5000.

3.3. Um estimador logístico de diferença de quartil mínimo:

Nosso terceiro exemplo é uma versão do estimador LQD usado no multinomRob. Usando a função R IQR para calcular o intervalo interquartil, a função a ser minimizada pode ser definida como segue.10

```
> LQDxmpl <- função(b) { + logística
<- função(x) { 1/(1+exp(-x)) }
```

10O problema de LQD resolvido no multinomRob é um pouco diferente. Aí o problema é minimizar a estatística de $\frac{hK}{2}$ ordem do conjunto de diferenças absolutas entre os resíduos padronizados, onde hK é uma função do tamanho da amostra e do número de coeficientes desconhecidos do modelo de regressão (Mebane e Sekhon 2004). O problema considerado no exemplo atual é mais simples, mas apresenta dificuldades de estimativa semelhantes.

	genoud				
	pop.size=5000a fun .		pop.size=10000a		
max=30		máx = 100	max=30	max=100 FEPb	
1	1.453e-32	1,658e-32	2,416e-32	6,134e-32	5,7e-4
2	6.55e-16	7.212e-16	9.652e-16	1.043e-15	8.1e-3
3	4.633e-18	3.918e-18	3.787e-18	4.032e-18	1.6e-2
4	6.203e-17	6.542e-17	9.453e-17	7.85e-17	0,3
5	0,07973	5.887e-08	8.133e-08	8.917e-08	5.06
6	18,58	0,08	9,38	-12569,49	-12569,49
8	-12569,49	-12569,49	-12554,5		
9	2,786	0	0,9353	0	4.6e-2
10	2,849	3.997e-15	2.199	3.997e-15	1.8e-2
11	7.994e-17	7.105e-17	9.548e-17	6.439e-17	1.6e-2
12	5,371e-19	0,004147	0,002073	1.178e-19	9.2e-6
13	0,02095	0,006543	0,006629	0,003011	1.6e-4
14	0,998	0,998	0,998	0,998	1,22
15	0,0003441	0,0004746	0,0003807	0,0003807	5,0e-4
16	-1,0316285	-1,0316285	-1,0316285	-1,0316285	-1,03
17	0,3979	0,3979	0,3979	0,3979	0,398
18	3	3	3	3	3.02
19	-3,863	-3,863	-3,863	-3,863	-3,86
20	-3,274	-3,277	-3,279	-3,286	-3,27
21	-9,85	-9,444	-9,95	-10,05	-5,52
22	-9.771	-10.09	-10.19	-10,3	-5,52
23	-10,1	-9.997	-10,32	-10.21	-6,57

Nota: bValores médios mínimos da função (acima de 50 repetições) obtidos usando genoud.
aMean melhores valores de função (mais de 50 replicações) relatados para a evolução "rápida"
algoritmo de programação, de Yao et al. (1999, 85 e 88, Tabelas II–IV).

Tabela 3: Valores médios de 22 funções otimizadas

	genoud				
	pop.size=5000a		pop.size=10000a		
fun.	max=30	max=100	max=30	max=100	FEPb
1	9.997e-32	7.059e-32	9.562e-32	2.1e-31	1.3e-4
2	1.668e-15	1.621e-15	2.116e-15	2.102e-15	7.7e-4
3	4.568e-18	3.342e-18	4.38e-18	5.136e-18	1.4e-2
4	1,793e-16	1,758e-16	2,055e-16	2,002e-16	0,5
5	0,5638	4.921e-08	5.573e-08	4.955e-08	5.87
6	5,65	0,274	3.528	0	0
8	3,749e-10	1,071e-12	8,948e-09	6,365e-13	52,6
	1,864	0	1,179	1,2e-2	0
9 10	0,7146	0	0,702	2,1e-3	0
11	1.209e-16	1.582e-16	1.289e-16	8.713e-17	2.2e-2
12	2,336e-18	0,02052	7,423e-19	3,01e-6	0,666
13	0,03427	0,006867	0,006903	0,001508	7,3e-5
14	5.638e-12	8.894e-11	1.029e-12	4.35e-12	0,56
15	0,0001813	0,0003546	0,000251	0,0002509	3,2e-4
16	1.315e-14	9.751e-15	1.233e-14	1.054e-14	4.9e-7
17	5.422e-15	5.51e-15	4.925e-15	1.392e-14	1.5e-7
18	1,509e-13	3,477e-14	6,18e-14	2,907e-14	0,11
19	7.349e-15	1.521e-15	1.344e-14	7.255e-15	1.4e-5
20	0,05884	0,0583	0,05765	0,05504	5.9e-2
21	1.212	1,776	1,005	0,7145	1,59
22	1.937	1.269	1.052	0,7459	2.12
23	1.479	1.636	1.066	1,29	3.14

Nota: aDesvio padrão dos valores mínimos da função (acima de 50 repetições) obtidos usando genoud. bDesvio padrão dos melhores valores de função (mais de 50 repetições) relatado para o algoritmo de programação evolutiva “rápido”, de Yao et al. (1999, 85 e 88, Tabelas II-IV).

Tabela 4: Desvios padrão de valores de 22 funções otimizadas

```
+ sIQR <- função(y, yhat, n) {
+ IQR((y-yhat)/sqrt(yhat*(n-yhat)), na.rm=TRUE)
+ }
+ sIQR(y, m*logística(x %*% b), m)
+ }
```

Para este exemplo, definimos LQDxmpl após calcularmos os dados simulados, de modo que o vetor de dados y, matriz x e escalar m estão no escopo para avaliar para ter os valores que simulamos:

```
> m <- 100
> x <- cbind(1,rnorm(1000),rnorm(1000))
> b1 <- c(.5, 1, -1)
> b2 <- c(0, -1, 1)
> logística <- função(x) { 1/(1+exp(-x)) }
> y <- rbinom(1000, m, logistic(c(x[1:900,] %*% b1, x[901:1000,] %*% b2)))
```

Os dados simulam contaminação substancial. As primeiras 900 observações são geradas por um modelo de regressão binomial enquanto as últimas 100 observações vêm de um modelo muito diferente.

Presumindo que estamos interessados no modelo que gerou a maior parte dos dados, ignorando a contaminação em um modelo linear generalizado com a família binomial produz resultados:

```
> resumo(glm1 <- glm(cbind(y,my) ~ x[,2] + x[,3], family="binomial"))

Ligar:
glm(formula = cbind(y, m - y) ~ x[, 2] + x[, 3], family = "binomial")

Resíduos de desvio:
     Mín.       1T   Mediana       3T      Máx.
-22,9168 -1,1693   0,3975  1,5895 24,6439

Coeficientes:
             Estimativa padrão Valor z do erro Pr(>|z|)
(Interceptar) 0,439492 0,007097 61,93 <2e-16 ***
x[, 2] 0,679847 0,007985 85,14 <2e-16 ***
x[, 3]       -0,716963 0,007852 -91,31 <2e-16 ***
---
Signif. códigos: 0 '***' 0,001 '**' 0,01 '*' 0,05 '.' 0,1' ' 1
```

Claro, se soubéssemos quais observações omitir os resultados seria muito melhor:

```
> suby <- y[1:900]
> subx <- x[1:900,]
> resumo(glm2 <- glm(cbind(suby,m-suby) ~ subx[,2] + subx[,3], family="binomial"))

Ligar:
glm(formula = cbind(suby, m - suby) ~ subx[, 2] + subx[, 3],
```

```
        família = "binomial")

Resíduos de desvio:
        Mín.       1T     Mediana       3Q      Máx.
-3,21478 -0,71699 0,03528 0,67867 2,88314

Coeficientes:
                    Estimativa padrão Valor z do erro Pr(>|z|)
(Interceptar) 0,501880 0,008036 62,46 <2e-16 ***
subx[, 2]           1,003592 0,009779 102,63 <2e-16 ***
subx[, 3] -0,984295 0,009437 -104,30 <2e-16 ***
---
Signif. códigos: 0 '***' 0,001 '**' 0,01 '*' 0,05 '.' 0,1' ' 1
```

Mas em aplicações práticas é desconhecido a priori quais observações devem ser tratadas como valores atípicos.

Como indica a definição de LQDxmpl, o LQD é baseado na minimização do intervalo interquartil (IQR) dos resíduos padronizados. Porque os quartis correspondem a diferentes pontos de dados para valores diferentes dos coeficientes de regressão, a função de aptidão não é suave, o que é dizer que não é em toda parte diferenciável. Em geral, em cada ponto do espaço de parâmetros onde a identidade do primeiro ou terceiro ponto quartil muda, a função não é diferenciável. A Figura 2 ilustra isso. Uma versão de alta resolução desta figura está disponível em Mebane e Sekhon (2011)—ver também http://sekhon.berkeley.edu/papers/rgenoudJSS.pdf.

O IQR claramente tem um mínimo na vizinhança dos valores dos coeficientes usados para gerar a maioria dos dados. Mas gráficos de contorno para a derivada parcial avaliada numericamente em relação ao segundo parâmetro do coeficiente testemunham a irregularidade local da função. A função que usamos para calcular esta derivada numérica é definida como segue.

```
> dLQDxmpl <- função(b) {
+ eps <- 1e-10
+ logística <- função(x) { 1/(1+exp(-x)) }
+ sIQR <- função(y, yhat, n) {
+           IQR((y-yhat)/sqrt(yhat*(n-yhat)), na.rm=TRUE)
+ }
+ dsIQR <- vetor()
+ for (i em 1:comprimento(b)) {
+         beijos <- b
+           beps[i] <- b[i]+eps
+           dsIQR <-
+               c(dsIQR,
+                   (sIQR(y, m*logístico(x %*% beps), m)-
+                       sIQR(y, m*logístico(x %*% b), m))/eps)
+ }
+ return(dsIQR)
+ }
```

Definindo a interceptação igual a 0,5, o código para gerar os valores plotados é

```
> blen <- 3 >
lenbseq <- comprimento(bseq <- seq(-2,2,comprimento=200)) >
bseq3 <- seq(-1.2,-.9,comprimento=200) > bseq2 <- seq(.
89,1,1,comprimento=200)
> IQRarr <- IQRarrA <- array(NA, dim=c((1+blen), lenbseq, lenbseq)) > dimnames(IQRarrA)
<- list(NULL, as.character(bseq), as.character(bseq) ) > dimnames(IQRarr) <- list(NULL,
as.character(bseq2), as.character(bseq3)) > for (i em 1:lenbseq) { + for (j em 1:lenbseq) {


+       IQRarrA[1,i,j] <- LQDxmpl(c(.5, bseq[i], bseq[j]))
+         IQRarrA[-1,i,j] <- dLQDxmpl(c(.5, bseq[i], bseq[j]))
+         IQRarr[1,i,j] <- LQDxmpl(c(.5, bseq2[i], bseq3[j]))
+         IQRarr[-1,i,j] <- dLQDxmpl(c(.5, bseq2[i], bseq3[j]))
+ } + }
```

O código a seguir produz os gráficos:

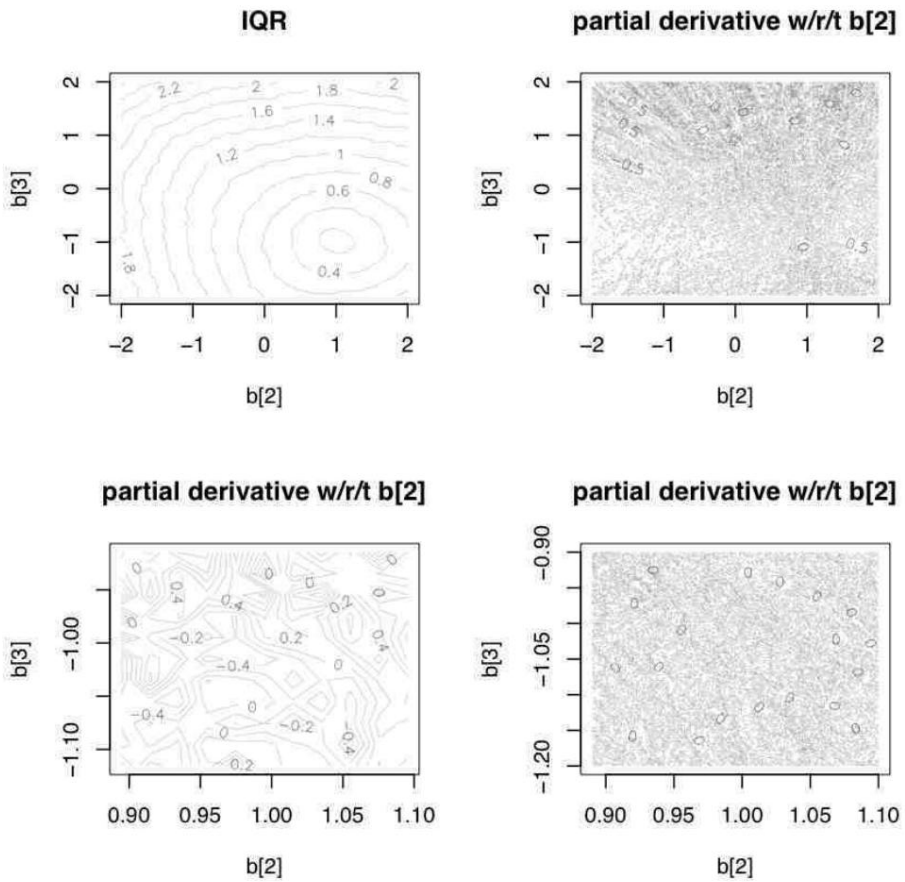
```
> par(mfrow=c(2,2), lwd=.1) >
contorno(bseq,bseq, IQRarrA[1,,], main="IQR", xlab="b[2]", ylab="b [3]") > contorno(bseq,bseq,
IQRarrA[3,,], main="derivada parcial w/r/tb[2]", xlab="b[2]", ylab="b[3] ") > loc2 <- (150:160)-5 > loc3
+               <- (40:50)+5 > contorno(bseq[loc2],bseq[loc3], IQRarrA[3,loc2,loc3],


+               main="derivada parcial w/r/tb[2]", xlab="b[2]",
+               ylab="b[3]")
> contorno(bseq2,bseq3, IQRarr[3,,], main="derivada parcial w/r/tb[2]",
+               xlab="b[2]", ylab="b[3]")
```

Se a função IQR fosse suave, veríamos linhas de contorno claramente separadas e levemente curvas, refletindo a não linearidade da função logística, mas não há nada disso. Em vez disso, olhando sobre o domínio [ÿ2, 2]2 para o segundo e terceiro parâmetros de coeficiente de regressão, com 200 pontos avaliados ao longo de cada eixo (o gráfico superior direito), há uma nuvem manchada.
Isso reflete o fato de que a derivada muda de sinal com muita frequência ao longo do domínio: dos 40.000 pontos em que a derivada é avaliada, ela é positiva em 12.460 pontos e negativa em 27.540 pontos.

A função de aptidão LQD não é sensivelmente mais suave perto dos valores verdadeiros. Os dois gráficos inferiores mostram as derivadas parciais em relação ao segundo parâmetro de coeficiente avaliado sobre o domínio [.89, 1.1] × [ÿ1.2, ÿ.9]. O gráfico inferior esquerdo avalia as derivadas em 11 pontos ao longo de cada eixo, enquanto o gráfico inferior direito usa 200 pontos ao longo de cada eixo. No gráfico da esquerda é mais fácil ver a complexidade do particionamento do espaço de parâmetros à medida que a identidade do primeiro ou terceiro quartil muda. O gráfico inferior direito mostra a complexidade de fato replicada no grão mais fino das avaliações mais detalhadas (para ver isso, provavelmente é necessário ampliar o gráfico enquanto visualiza on-line a versão do Journal of Statistical Software desta documentação).11 Dentro desse domínio menor o sinal da derivada muda mesmo

11Consulte Mebane e Sekhon (2011) ou http://sekhon.berkeley.edu/papers/rgenoudJSS.pdf.

Figura 2: Gráficos de Contorno da Função de Aptidão LQD e de suas Derivadas Parciais



mais frequentemente do que no domínio [ÿ2, 2]2 : a derivada é positiva em 18.098 pontos e negativa em 21.902 pontos.

Embora a função de aptidão LQD possa ser diferenciável em uma vizinhança da solução global, essa vizinhança, se existir, claramente não é muito grande. O mais provável é que a solução global esteja localizada em um ponto em que a função não seja diferenciável. Portanto, um gradiente avaliado numericamente não é significativo para avaliar se a solução foi encontrada. Na solução verdadeira, os valores numéricos do gradiente podem diferir substancialmente de zero.

Para usar o LQD para estimar os parâmetros do modelo de regressão binomial, usamos a seguinte chamada para genoud. Como as informações de gradiente são de relevância questionável para esse problema, desativamos a condição de término de que o gradiente na solução seja menor que o valor especificado para o argumento solution.tolerance. Mantemos a configuração padrão BFGS=TRUE porque, em princípio, a otimização orientada por gradiente pode ajudar em cada uma das muitas vizinhanças diferenciáveis, mesmo que seja inútil nos limites não diferenciáveis. Nossa experiência otimizando o LQD (Mebane e Sekhon 2004) mostra que usar o BFGS dessa forma melhora o desempenho, mesmo que o gradiente não seja útil para avaliar se a solução foi encontrada.

```
> LQD1 <-
+ genoud(LQDxmpl, nvars=3, max=FALSE, pop.size=2000, max.generations=300,
+                 wait.generations=100, gradient.check=FALSE, print=1)
```

Esta invocação da função LQDxmpl corresponde ao comportamento de multinomRob na medida em que produz uma estimativa para o parâmetro de interceptação junto com os outros coeficientes. Em um contexto de regressão linear, a estatística de intervalo interquartil não contém informações sobre o interceptar, então o LQD não é um estimador para esse parâmetro. Com uma regressão binomial modelo existe alguma informação sobre o intercepto devido à não linearidade da logística função. A estimativa de LQD para a interceptação deve, no entanto, não ser muito Boa.

Os resultados da estimativa anterior são os seguintes.

```
Ter, 7 de agosto 02:27:08 2007
Domínios:
  -1,000000e+01 <= X1 <=                1.000000e+01
  -1,000000e+01 <= X2 <=                1.000000e+01
  -1,000000e+01 <= X3 <=                1.000000e+01

 [...]

HARD Número Máximo de Gerações: 300
Máximo de Gerações Inalteráveis: 100
Tamanho da população: 2000
Tolerância de convergência: 1.000000e-06

Usando o otimizador baseado em derivativos BFGS no melhor indivíduo de cada geração.
Não verificar gradientes antes de parar.
Usando Indivíduos Fora dos Limites.

Problema de minimização.


Geração#                    Valor da solução

       0                4.951849e-01
      56                1.298922e-01
      57                1.298891e-01
      59                1.298848e-01
      60                1.298820e-01
      61                1.298793e-01
      62                1.298768e-01
      63                1.298744e-01

 limite de 'wait.generations' atingido.
Nenhuma melhoria significativa em 100 gerações.
```

Valor de adequação da solução: 1,298743e-01

Parâmetros na Solução (parâmetro, gradiente):

X[1]:	8.130357e-02	G[1]:	9.616492e-03
X[2]:	8.889485e-01	G[2]:	-1.167897e-01
X[3]:	-9.327966e-01	G[3]:	-3.090130e-02

Solução encontrada geração 63

Número de Gerações Executadas 164

Ter, 7 de agosto 02:31:09 2007

Tempo total de execução: 0 horas 4 minutos e 1 segundos

Lembre-se de que o gradiente não é informativo de forma confiável na solução. Para verificar se este solução for crível, podemos tentar reestimar o modelo usando uma população maior e maior número especificado de gerações:

```
> LQD1 <-
+ genoud(LQDxmpl, nvars=3, max=FALSE, pop.size=10000, max.generations=1000,
+                 wait.generations=300, gradient.check=FALSE, print=1)
```

Ao preço de um tempo de execução muito maior (de quatro minutos até uma hora e 53 minutos), os resultados são melhores do que a primeira execução (mesmo que a medida resumida de ajuste seja ligeiramente pior):

Problema de minimização.

Geração#	Valor da solução
0	2.238865e-01
2	1.301149e-01
3	1.300544e-01
4	1.300482e-01
6	1.300375e-01
7	1.300343e-01
8	1.300323e-01
134	1.299662e-01
135	1.299099e-01
136	1.298867e-01
137	1.298843e-01
138	1.298822e-01
139	1.298791e-01
141	1.298774e-01

limite de 'wait.generations' atingido.

Nenhuma melhoria significativa em 300 gerações.

Valor de adequação da solução: 1,298770e-01

Parâmetros na Solução (parâmetro, gradiente):

X[1]:	2.013748e-01	G[1]:	-7.394125e-02
X[2]:	9.526390e-01	G[2]:	7.807607e-02
X[3]:	-9.642458e-01	G[3]:	3.834052e-02

Solução encontrada Geração 141

Número de Gerações Executadas 442

Ter, 7 de agosto 05:16:37 2007

Tempo total de execução: 1 hora 53 minutos e 45 segundos

Este exemplo demonstra uma dificuldade chave que surge ao otimizar funções irregulares na ausência de gradientes. É difícil avaliar quando ou se um ótimo foi encontrado. Os valores dos coeficientes estimados são próximos aos usados para gerar a maioria dos dados, exceto como esperado a estimativa para o intercepto não é boa. As estimativas são melhores do que se tivéssemos ignorado a possibilidade de contaminação. Mas se essas são as melhores estimativas possíveis não está claro. Se usarmos uma população ainda maior e especificarmos que um número ainda maior de gerações seja executado, talvez uma solução melhor seja encontrada.

Mesmo para problemas menos irregulares, a convergência é difícil de determinar. Os otimizadores não lineares geralmente relatam falsa convergência e os usuários não devem simplesmente confiar em qualquer critério de convergência que um otimizador use. McCullough e Vinod (2003) oferecem quatro critérios para verificar a solução de um solver não linear. Esses critérios são significativos apenas para problemas que atendem às condições de regularidade na solução, notadamente a diferenciabilidade e, como tal, não são úteis para o exemplo LQD oferecido acima. Os quatro critérios são: (1) garantir que os gradientes sejam zero; (2) inspecionar o caminho da solução (ou seja, o traço) para certificar-se de que segue a taxa de convergência esperada; (3) avaliar a Hessiana para certificar-se de que está bem condicionada;12 e (4) traçar o perfil da probabilidade de verificar se uma aproximação quadrática é adequada. Pode ser necessário obter os resultados do genoud e passá-los para o nível ótimo para realizar alguns desses testes de diagnóstico, como o perfil da probabilidade. Também é uma boa prática usar mais de um otimizador para verificar a solução (Stokes 2004).

Observe que o genoud usa suas próprias sementes de números aleatórios e geradores de números pseudoaleatórios internos para garantir a compatibilidade com a versão C original do software e tornar o comportamento do cluster mais previsível. Essas sementes são definidas pelas opções unif.seed e int.seed. O comando R set.seed é ignorado pelo genoud.

4. Conclusão

12Após uma troca com Drukker e Wiggins (2004), McCullough e Vinod (2004a) modificam sua terceira sugestão para observar que determinar se o Hessian está bem condicionado depende em parte de como os dados são dimensionados. Ou seja, um Hessian que parece estar mal condicionado pode ficar bem condicionado por reescalonamento.
Portanto, se uma Hessiana parece estar mal condicionada, McCullough e Vinod (2004a) recomendam que o analista tente determinar se o reescalonamento dos dados pode resultar em uma Hessiana bem condicionada.

A função genoud oferece muito mais opções do que podem ser revisadas neste breve artigo.
Essas opções são descritas no arquivo de ajuda do R. A opção mais importante que influencia o quão bem o algoritmo evolucionário funciona é o argumento pop.size. Este argumento controla o tamanho da população – ou seja, é o número de indivíduos que genoud usa para resolver o problema de otimização. Como observado acima, os teoremas que provam que algoritmos genéticos encontram boas soluções são assintóticos tanto no tamanho da população quanto no número de gerações. Portanto, é importante que o valor pop.size não seja pequeno. Por outro lado, o tempo computacional é finito, portanto, compensações óbvias devem ser feitas. Como o exemplo LQD ilustra, um tamanho populacional maior não é necessariamente melhor. As opções mais importantes para garantir que uma boa solução seja encontrada, além de pop.size, são wait.generations, max.generations e hard.generation.limit.

Muitos modelos estatísticos têm funções objetivas que são funções não lineares dos parâmetros, e otimizar tais funções é um negócio complicado (Altman, Gill e McDonald 2003).
Dificuldades de otimização muitas vezes surgem mesmo para problemas que são geralmente considerados simples. Para uma história de advertência sobre como a otimização pode ser uma tarefa difícil mesmo para tais problemas, veja o esforço de Stokes (2004) para replicar um modelo probit estimado por Maddala (1992, pp. 335). Uma controvérsia recente sobre a estimativa de um modelo não linear estimado pela máxima verossimilhança oferece outro conto de advertência (Drukker e Wiggins 2004; McCullough e Vinod 2003, 2004b,a; Shachar e Nalebuff 2004). Os usuários finais geralmente ficam surpresos ao saber que esses problemas de otimização podem surgir e que os resultados podem variar substancialmente entre otimizadores e implementações de software.

O pacote rgenoud fornece um otimizador global poderoso e flexível. Quando comparado com métodos tradicionais de otimização baseados em derivativos, o rgenoud tem um bom desempenho (Sekhon e Mebane 1998). A otimização de funções irregulares é, no entanto, tanto uma arte quanto uma ciência.
E um otimizador não pode ser usado sem pensar, mesmo para superfícies simples, muito menos espaços que requerem um algoritmo genético. Esperamos que a disponibilidade de um otimizador global escalável permita que os usuários trabalhem com funções difíceis de forma mais eficaz.

Referências

Abadie A, Diamond A, Hainmueller J (2011). “Sintetizador: Um Pacote R para Métodos de Controle Sintético em Estudos de Caso Comparativos.” Jornal de software estatístico. Em breve, URL http://www.jstatsoft.org/.

Abadie A, Gardeazabal J (2003). “Os custos econômicos do conflito: um estudo de caso-controle para o País Basco”. American Economic Review, 92(1).

Altman M, Gill J, McDonald M (2003). Questões Numéricas em Computação Estatística para o Cientista social. John Wiley and Sons, Nova York.

Altman M, McDonald MP (2011). “BARD: Redistritamento Melhor Automatizado.” Diário de Software Estatístico. Em breve, URL http://www.jstatsoft.org/.

Billingsley P (1986). Probabilidade e Medida. Wiley, Nova York.

Braumoeller BF (2003). “Complexidade causal e o estudo da política”. Análise Política, 11(3), 209-233.

Braumoeller BF, Goodrich B, Kline J (2010). Boolean: Modelos Booleanos de Resposta Binária. Pacote R versão 2.0-2, URL http://polisci.osu.edu/faculty/braumoeller/custom/Booleanbatch.html .

Davis L (ed.) (1991). Manual de Algoritmos Genéticos. Van Nostrand Reinhold, Nova York.

De Jong KA (1993). “Algoritmos genéticos não são otimizadores de função.” Em LD Whitley (ed.), Fundamentos de Algoritmos Genéticos 2. Morgan Kaufmann, San Mateo, CA.

Diamond A, Hainmueller J (2008). Sintetizador: Método de Grupo de Controle Sintético para Estudos de Caso Comparativos. Pacote R versão 0.1-6, URL http://CRAN.R-project.org/package= Synth.

Drukker DM, Wiggins V (2004). “Verificando a solução de um solucionador não linear: um caso Estudo: Comente.” American Economic Review, 94(1), 397-399.

Efron B, Tibshirani RJ (1994). Uma introdução ao Bootstrap. Chapman & Hall, Novo Iorque.

Feller W (1970). Uma introdução a teoria da probabilidade e as suas aplicações. Wiley, Novo Iorque. Vol. 1, 3d ed., impressão revisada.

Filho José L Ribeiro PCT, Alippi C (1994). “Ambientes de Programação de Algoritmos Genéticos”. Computador, 27, 28-43.

Gill PE, Murray W, Wright MH (1981). Otimização prática. Imprensa Acadêmica, San Diego.

Ginsburger D, Roustant O (2010). DiceOptim: Otimização baseada em krigagem para experimentos em computador. Pacote R versão 1.0, URL http://cran.r-project.org/package= DiceOptim.

Goldberg DE (1989). Algoritmos Genéticos em Busca, Otimização e Aprendizado de Máquina. Addison-Wesley, Reading, MA.

Goodrich B (2011). FAiR: Análise Fatorial no pacote R.R versão 0.4-7, URL http: //cran.r-project.org/package=FAiR.

Grefenstette JJ (1993). “Engano Considerado Nocivo.” Em LD Whitley (ed.), Fundações de Algoritmos Genéticos 2. Morgan Kaufmann, San Mateo, CA.

Grefenstette JJ, Baker JE (1989). “Como funcionam os algoritmos genéticos: um olhar crítico sobre o paralelismo implícito”. In Proceedings of the Third International Conference on Genetic Algorithms, pp. 20–27. Morgan Kaufmann, San Mateo, CA.

Holanda JH (1975). Adaptação em sistemas naturais e artificiais. Universidade de Michigan Imprensa, Ann Arbor.

Rei G, Varinha J (2007). “Comparando respostas de pesquisa incomparáveis: avaliando e selecionando vinhetas de ancoragem.” Análise Política, 15, 46-66.

Kuhnt S, Rudak N (2011). JOP: Gráfico de Otimização Conjunta. Pacote R versão 2.0.1, URL http://cran.r-project.org/package=JOP.

Lee CY, Lee YJ (2009a). PKfit: Uma Ferramenta para Análise de Dados em Farmacocinética. Pacote R versão 1.1.8, URL http://CRAN.R-project.org/package=PKfit.

Lee HY, Lee YJ (2009b). ivivc: modelagem de correlação in vitro-in vivo (IVIVC). Pacote R versão 0.1.5, URL http://CRAN.R-project.org/package=ivivc.

Maddala G (1992). Introdução à Econometria. 2ª edição. MacMillan, Nova York.

McCullough BD, Vinod HD (2003). “Verificando a solução de um solucionador não linear: um caso Estudar." American Economic Review, 93(3), 873-892.

McCullough BD, Vinod HD (2004a). “Verificando a solução de um solucionador não linear: um estudo de caso: resposta.” American Economic Review, 94(1), 400-403.

McCullough BD, Vinod HD (2004b). “Verificando a solução de um solucionador não linear: um estudo de caso: resposta.” American Economic Review, 94(1), 391-396.

Mebane Jr WR, Sekhon JS (1997). Otimização GENÉTICA Usando Derivados (GENOUD). Programa de computador disponível mediante solicitação.

Mebane Jr WR, Sekhon JS (2004). “Estimativa robusta e detecção de valores discrepantes para modelos multinomiais superdispersos de dados de contagem.” American Journal of Political Science, 48(2), 391–410.

Mebane Jr WR, Sekhon JS (2009). multinomRob: Estimação robusta de modelos de regressão multinomial superdispersos. Pacote R versão 1.8-4, URL http://CRAN.R-project. org/package=multinomRob.

Mebane Jr WR, Sekhon JS (2011). “Otimização genética usando derivativos: o pacote rge noud para R.” Jornal de software estatístico. Em breve, URL http://www. jstatsoft.org/.

Michalewicz Z (1992). Algoritmos Genéticos + Estruturas de Dados = Programas de Evolução. Springer Verlag, Nova York.

Michalewicz Z, Swaminathan S, Logan TD (1993). “GENOCOP (versão 2.0).” Código fonte do programa de computador em linguagem C. http://www.cs.adelaide.edu.au/~zbyszek/EvolSyst/genocop2.0.tar.Z .

Nix AE, Vose MD (1992). “Modelando Algoritmos Genéticos com Cadeias de Markov.” Anais de Matemática e Inteligência Artificial, 5, 79-88.

Picheny V, Ginsburger D (2011). KrigInv: Inversão baseada em krigagem para experimentos de computador determinísticos e barulhentos. Pacote R versão 1.1, URL http://CRAN.R-project.org/ package=KrigInv.

R Development Core Team (2009). R: Uma Linguagem e Ambiente para Computação Estatística. R Foundation for Statistical Computing, Viena, Áustria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sekhon JS (2011a). “Software de correspondência de pontuação multivariada e de propensão com otimização automatizada de saldo: o pacote de correspondência para R.” Jornal de software estatístico. Em breve, URL http://www.jstatsoft.org/.

Sekhon JS (2011b). Correspondência: Correspondência multivariada e de pontuação de propensão com pesquisa automatizada de saldo. Pacote R versão 4.7-12, URL http://CRAN.R-project.org/package=Matching.

Sekhon JS, Mebane Jr WR (1998). “Otimização Genética Usando Derivativos: Teoria e Aplicação a Modelos Não Lineares.” Análise Política, 7, 189-203.

Shachar R, Nalebuff B (2004). “Verificando a solução de um solucionador não linear: um estudo de caso: comentário.” American Economic Review, 94(1), 382-390.

Stokes H (2004). “Sobre a vantagem de usar dois ou mais sistemas de software econométricos para resolver o mesmo problema.” Journal of Economic and Social Measurement, 29(1-3), 307–320.

Vose MD (1993). “Modelando Algoritmos Genéticos Simples”. Em LD Whitley (ed.), Fundações de Algoritmos Genéticos 2. Morgan Kaufmann, San Mateo, CA.

Varinha J, Rei G (2011). “Âncoras: Software para ancoragem de dados de vinheta”. Diário de Software Estatístico. Em breve, URL http://www.jstatsoft.org/.

Wand J, King G, Lau O (2008). Âncoras: Software para ancoragem de dados de vinheta. Pacote R versão 2.0, URL http://wand.stanford.edu/anchors/.

Yao X, Liu Y, Lin G (1999). “Programação Evolucionária Mais Rápida.” IEEE Transactions on Evolutionary Computation, 3(2), 82–102.

Afiliação:

Walter R. Mebane, Jr.
Departamento de Ciência Política
Departamento de Estatística
Universidade de Michigan Ann
Arbor, MI 48109-1045 E-mail:
wmebane@umich.edu URL: http://
www.umich.edu/~wmebane

Jasjeet S. Sekhon
Departamento de Ciência Política
Departamento de Estatística 210
Barrows Hall #1950 UC Berkeley
Berkeley, CA 94720-1950 E-mail:
sekhon@berkeley.edu URL: http://
sekhon.berkeley.edu