Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Sociais Aplicadas
Graduação em Economia

Soluções numéricas para o problema de principal-agente

José Bruno do Nascimento Clementino

Mariana, Minas Gerais
28 de novembro de 2019

José Bruno do Nascimento Clementino

Soluções numéricas para o problema de principal-agente

Monografia apresentada ao Curso de Ciências Econômicas do Instituto de Ciências Sociais Aplicadas (ICSA) da Universidade Federal de Ouro Preto (UFOP), como requisito à obtenção de grau de Bacharel em Ciências Econômicas

Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Sociais Aplicadas

Graduação em Economia

Orientador: Prof. Dr. Martin Harry Vargas Barrenechea

Mariana, Minas Gerais

28 de novembro de 2019

C626s Clementino, José Bruno do Nascimento.
Soluções numéricas para o problema de principal-agente [manuscrito] / José Bruno do Nascimento Clementino. - 2019.

57f.: il.: color; grafs; tabs.

Orientador: Prof. Dr. Martin Harry Vargas Barrenechea.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Sociais Aplicadas. Departamento de Ciências Econômicas e Gerenciais.

1. Teoria dos jogos - Teses. 2. Equações - Soluções numéricas - Teses. I. Barrenechea, Martin Harry Vargas. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 519.83

Catalogação: ficha.sisbin@ufop.edu.br

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS SOCIAIS E APLICADAS
DEPARTAMENTO DE CIÊNCIAS ECONÔMICAS

FOLHA DE APROVAÇÃO

José Bruno do Nascimento Clementino

Soluções numéricas para o problema de principal-agente

Membros da banca

Martin Harry Vargas Barrenechea - Prof. Dr. - UFOP
Antônio Francisco Neto - Prof. Dr. - UFOP
Marcelo Aparecido Cabral Nogueira - Prof. Dr. - UFOP

Versão final
Aprovado em 29 de Novembro de 2019

De acordo

Martin Harry Vargas Barrenechea

Documento assinado eletronicamente por **Martin Harry Vargas Barrenechea**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 29/11/2019, às 14:45, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0025024** e o código CRC **09C8F7B0**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: - www.ufop.br

Agradecimentos

Agradeço a Deus pela oportunidade de estar vivo. Agradeço também aos meus pais que deram seu suor e sangue para que eu pudesse concluir minha graduação; ao meu orientador Martin Barrenechea pela paciência e todas as oportunidades de aprendizado (e principalmente por me ensinar matemática básica); ao meu amigo Guilherme Nunes que me incentivou a buscar conhecimento; à Ana Paula que me acompanhou durante toda a trajetória; aos meus aliados da "Turma"; e à república Largados que me acolheu durante a maior parte do curso.

“Never let the future disturb you.
You will meet it, if you have to,
with the same weapons of reason
which today arm you against the present.
(Meditations, VII, 8)

Resumo

Dentro da teoria dos jogos, problemas de principal-agente ocorrem quando uma das partes interessadas num contrato qualquer detém informação relevante para o seu bom cumprimento. Essa classe de jogos já foi bem definida analiticamente, no entanto, observou-se uma lacuna com relação a possíveis soluções numéricas para esse tipo problema. Utilizando funções de penalidade e algoritmos de otimização implementados em R, problemas de seleção adversa e risco moral foram avaliados e comparados às implicações teóricas. Os resultados foram positivos e os métodos permitem que vários outros problemas de estrutura similar possam ser avaliados pelos algoritmos propostos.

Palavras-chave: problemas de agenciamento. métodos numéricos. teoria dos jogos.

Abstract

In game theory, principal-agent problems happen when one of the participants of a contract holds relevant information to its realization. This class of games is well defined analytically, although, it has been observed the lack of content related to numerical solutions. Through the use of penalty functions and optimization algorithms implemented in R, adverse selection and moral hazard problems were evaluated and compared to its theoretical implications. The results were positive and the methods allow that other problems with similar structure be evaluated by the proposed algorithms.

Keywords: agency theory. numerical methods. game theory.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Sumário

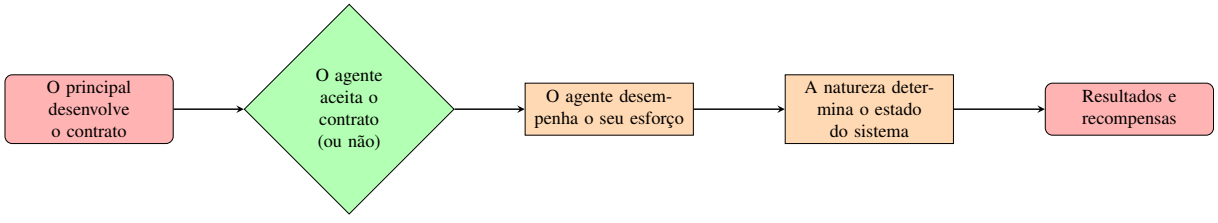
1 Introdução

Uma relação agente-principal é estabelecida quando um indivíduo (ou instituição) contrata um segundo indivíduo para que determinada tarefa seja executada. Por exemplo, um eleitorado que vota e elege um governo. Ou o dono de uma empresa que contrata um CEO para gerenciá-la. O principal é a parte contratante e o agente é a parte contratada. Esse relacionamento é estabelecido através de um contrato que especificará informações a respeito do trabalho que deve ser executado e o pagamento pelo sucesso ou não das tarefas.

A condição de firmamento do contrato entre principal e agente ocorre quando a utilidade do agente de assinar o contrato supera a utilidade de não fazê-lo. Porque os atores que participam dessa relação contratual possuem interesses distintos e, em algumas situações, uma das partes detém informação relevante para o cumprimento das cláusulas de um contrato ou até mesmo seu firmamento, o estabelecimento de relações contratuais implicam em situações de decisão sob incerteza.

Essas situações, com certa cautela, podem ser estruturadas como um jogo. Em cada período, os atores participantes dessa relação econômica, principal e agente, escolhem uma determinada estratégia levando em conta o que a outra vai fazer em cada período visando a maximização de seu bem-estar. No entanto, mesmo quando os atores tomam as melhores atitudes possíveis, os resultados podem não ser os esperados. Isso ocorre porque, imaginando uma situação hipotética de informação simétrica, a informação pode não ser perfeita, o que implica que o principal e o agente podem ter seu relacionamento afetado por um elemento aleatório. Nesse caso, a "natureza"é responsável por decidir algo pertinente à determinação do sucesso dos contratos. A figura 1 mostra a sequência.

Figura 1 – Cronograma de jogo com informação simétrica



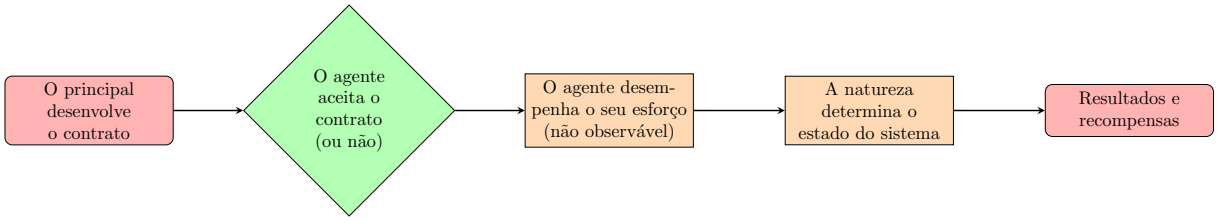
Fonte: Elaborada pelo próprio autor.

Os problemas tratados nesse trabalho assumem que a informação entre os atores da relação contratual não é simétrica. As consequências disso são o surgimento de ineficiências econômicas. A literatura trata de vários problemas que surgem de relações contratuais com informação simétrica. Duas delas são destacadas: a de *risco moral*, na qual a variável que

determina o sucesso de um contrato não é observável pelo principal, mas é observável pelo agente; e a de *seleção adversa*, na qual a variável de sucesso é observável, mas ela depende de um tipo de agente que não revela as informações necessárias.

Por exemplo, no caso de risco moral, um trabalhador pode não se dedicar o suficiente para que seu trabalho seja de qualidade, mas mesmo assim ele pode ter sorte e acabar obtendo sucesso. O empregador observa esse resultado, mas ele não consegue definir qual foi o esforço empregado na atividade. O cronograma da situação é descrito pela figura 2

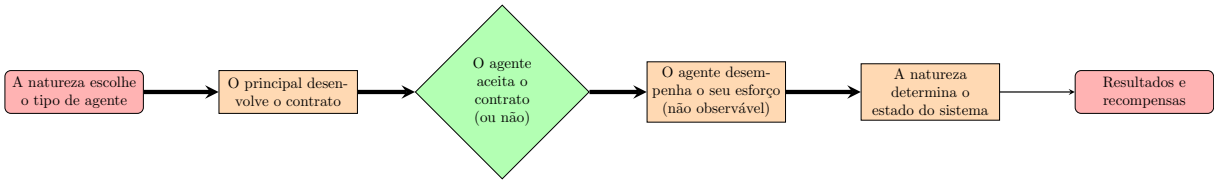
Figura 2 – Cronograma do problema de Risco Moral



Fonte: Elaborada pelo próprio autor

Para o caso da seleção adversa, numa relação entre um corretor de seguros e um cliente, o corretor de seguros pode oferecer uma apólice destinada a pessoas que não tem comportamento arriscado, mas o cliente pode ser o tipo de pessoa que toma altos riscos e essa informação não é revelada por ele. O cronograma da situação é descrito pela figura 3.

Figura 3 – Cronograma do problema de Seleção Adversa



Fonte: Elaborada pelo próprio autor.

As soluções para esses tipos de problemas advindos da informação assimétrica dependem da utilização de múltiplas restrições que restringem o número de contratos possíveis para aqueles que são eficientes. A literatura já proporcionou variadas soluções analíticas para esses tipos de problemas. Dada a dificuldade de se trabalhar com múltiplas restrições, o objetivo do trabalho é investigar a possibilidade de se empregar outra metodologia que não seja a analítica para analisar as características dos jogos em questão. Especificamente, a utilização de métodos computacionais para esse tipo de problema.

De acordo com Judd (1998), métodos computacionais são utilizados com o objetivo de resolver diversos tipos de problemas dentro da economia, como a utilização de algoritmos para computar equilíbrios de Nash em jogos, computar equilíbrios gerais, encontrar soluções

para modelos não-lineares estocásticos de expectativas racionais e outros mais. Sabendo disso, propõe-se, então, investigar uma solução computacional para dois problemas: um de risco moral e outro de seleção adversa e, então, investigar seus resultados. Métodos numéricos são comumente utilizados para a solução de problemas de engenharia, mas nenhum conteúdo em português que se propõe solucionar esse tipo de problema (principal-agente) foi encontrado.

No primeiro capítulo, uma revisão teórica acerca de assuntos relacionados ao problema de principal-agente é feita. No segundo capítulo, a metodologia empregada para solucionar e analisar os problemas é descrita. No terceiro capítulo, o processo de implementação computacional desses algoritmos é demonstrado, como apoio ao conteúdo presente nos apêndices A e B. No quarto capítulo, uma descrição dos resultados obtidos por meio da aplicação da metodologia é feita. O último capítulo, de considerações finais, trata das conclusões deste estudo, as limitações encontradas e possíveis extensões para o trabalho. Os apêndices garantem que todos os resultados encontrados sejam replicáveis para quem possa interessar.

2 Problemas de principal-agente

Neste capítulo, a formalização e solução analítica para os problemas de principal-agente são identificadas, bem como as ideias gerais necessárias para a construção do modelo. Um modelo matemático utilizado para a análise dos problemas de principal-agente é construído ao longo das seções.

2.1 Teoria dos jogos, decisão sob incerteza e contratos

2.1.1 Uma breve observação sobre a teoria dos jogos

De acordo com Gibbons (1992), a teoria dos jogos é um estudo de problemas que envolvem múltiplos agentes. Através dela, os resultados advindos da interação entre esses agentes podem ser analisados. Todo jogo possui jogadores que empregam ações distintas visando resultados específicos de acordo com suas preferências. Dentro da economia, várias situações podem ser estruturadas dessa maneira. Normalmente, como observado por Machina (1987, p. 124), os pressupostos com relação aos agentes são análogos ao caso da teoria do consumidor.

Um exemplo de jogo simples, desenvolvido por Fudenberg e Tirole (1991), envolve dois caçadores que devem decidir ao mesmo tempo se vão caçar um cervo ou uma lebre. Se ambos caçam cervo, eles capturam um e dividem por dois. Por outro lado, se eles caçam lebre, cada um deles capturam uma lebre. Por fim, se ambos decidem caçar separadamente, o que caçar o animal menor obtém sucesso, mas o que caça o maior fracassa. Assumindo que, por exemplo, o cervo valha 4 "utils"e a lebre valha 1 "util", um quadro que descreve a situação pode ser criado da seguinte maneira:

Quadro 1 – Jogo dos Caçadores

		Caçador 1	
		Lebre	Cervo
Caçador 2	Lebre	$(1,1)$	$(1,0)$
	Cervo	$(0,1)$	$(2,2)$

Fonte: Elaborado com base em Fudenberg e Tirole (1991)

Observa-se que a cooperação é o melhor resultado via equilíbrio de Nash, dado que ambos caçadores irão obter a maior utilidade possível quando agem dessa maneira. No entanto, se as recompensas fossem ligeiramente alteradas, com os pressupostos básicos utilizados, o equilíbrio seria diferente. Além disso, esse é um jogo de informação completa e estático, então, caso a informação fosse incompleta ou jogo se alterasse a cada período de tempo, pode-se

assumir que o resultado seria diferente também. Na teoria de agenciamento, a assimetria de informação é a causa dos problemas de principal-agente.

Quando a informação é incompleta ou imperfeita, os atores precisam tomar decisões com base nas crenças a respeito do que pode ter sido feito pelas partes envolvidas no decorrer de um jogo. Nessas situações, o ator que busca maximizar suas recompensas precisa ser capaz de tomar as melhores decisões possíveis utilizando as probabilidades de que certos eventos ocorram dado que outros eventos tenham ocorrido, ou seja, a regra de Bayes.

Os jogos estudados nesse trabalho levam em conta, além da assimetria de informação, uma sequencialidade de períodos. Nesses casos, de acordo com Gibbons (1992, p. 176), em cada período, discreto, um dos atores precisa tomar alguma decisão específica. Esses indivíduos necessitam ter uma crença do que foi feito pelas partes envolvidas e, a partir disso, é preciso conjecturar qual o estado do jogo após a tomada de decisão da outra parte envolvida. Conforme o jogo vai se desenvolvendo, todas as decisões tomadas precisam ser sequencialmente racionais, ou seja, dado um estado qualquer do jogo, as próximas decisões serão as melhores possíveis de acordo com as crenças dos atores envolvidos. Como as decisões tomadas são as melhores para cada estado e as crenças são conjecturadas a partir da regra de Bayes, os equilíbrios desses jogos são denominados equilíbrios bayesianos dinâmicos.

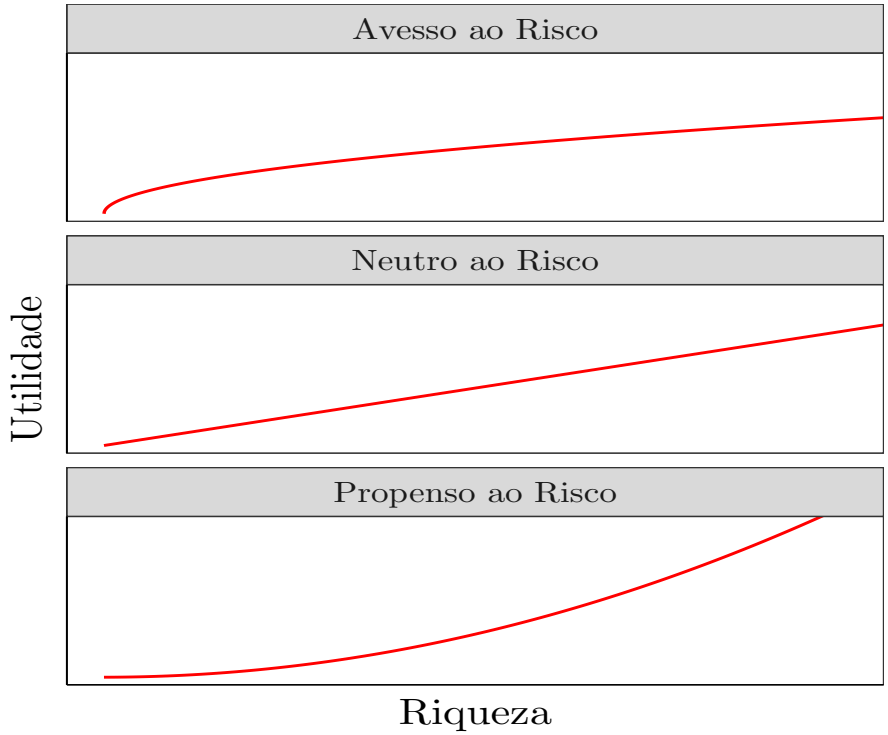
2.1.2 Utilidade esperada

Para tratar das preferências dos indivíduos com relação a situações de incerteza, a utilidade esperada pode ser utilizada. Ela define como indivíduos de uma relação qualquer reagem no que diz respeito ao risco. Uma das pressuposições desse modelo é a de que os agentes possuem funções de utilidade do tipo von Neumann-Morgenstern. Assim sendo, os agentes podem ser neutros, avessos ou propensos ao risco. De acordo com Varian (2014, p. 230), numa situação de aposta ou incerteza:

- O agente avesso ao risco, prefere manter sua riqueza esperada do que fazer uma aposta ou se arriscar;
- O agente propenso ao risco está mais disposto a fazer apostas do que manter sua riqueza;
- O agente neutro ao risco não está preocupado com os possíveis riscos de apostar sua riqueza, somente com o valor esperado de sua riqueza após apostas.

O formato das curvas de utilidade para cada agente é distinto para cada tipo de comportamento. Ser avesso ao risco torna a função de utilidade mais côncava, enquanto ser propenso ao risco a torna mais convexa. No caso da neutralidade com relação ao risco, a função é linear. A figura 4 descreve as relações visualmente.

Figura 4 – Função de utilidade do agente e seu comportamento em relação ao risco



Fonte: Elaborada com base em Varian (2014).

2.1.3 Contratos

Em situações de incerteza, quando agentes buscam entre si o estabelecimento de uma transação, ambas as partes podem formalizar um contrato de modo a estabelecer as atividades que devem ser cumpridas entre os indivíduos (ou instituições) participantes e os parâmetros de sucesso. Contratos podem definir incentivos que levem as partes a terem comportamentos específicos. No entanto, de acordo com Fiani (2012), a complexidade, a incerteza e a racionalidade limitada dos agentes (sendo um determinante possível fatores neurofisiológicos) tornam difícil o estabelecimento de cláusulas contratuais para cada resultado possível decorrente de uma transação.

No processo de criação, o responsável por elaborar um contrato pode não ter informações suficientes para estabelecer cláusulas que prevejam todos os possíveis resultados das ações do contratado e, por essa razão, o último pode tomar alguma atitude oportunista, no sentido de que se utiliza de possíveis falhas nas condições propostas do contrato em seu próprio benefício. Segundo Macho-Stadler e Pérez-Castrillo (1997), de modo a contornar tal comportamento sob situações que envolvam incerteza, contratos são elaborados de modo a promover incentivos para que as ações mais eficientes possíveis sejam tomadas ou para que informação privada de difícil obtenção em outras circunstâncias seja obtida. Em ambas as ocasiões, o objetivo é tornar o resultado do contrato eficiente no sentido de Pareto para as partes contempladas.

Contratos podem ter diferentes formatos. Fiani (2012) descreve quatro tipos básicos, sendo eles:

- Contratos que estabelecem no presente um resultado específico no futuro;
- Contratos que estabelecem no presente um resultado específico no futuro, e que inclui em suas condições a possibilidade de outros resultados antecipados previamente ocorrerem, denominados contratos de cláusulas condicionais;
- Contratos de duração rápida, que só são executados quando as condições necessárias para a efetivação de uma transação ocorrem de fato, denominado contratos de curto prazo sequencial;
- Contratos estabelecidos no presente em que o contratante reserva para si o direito de selecionar ações específicas de um agente dentre várias outras ações possíveis estipuladas anteriormente, denominados relações de autoridade.

2.2 O problema de risco moral

O problema é exposto aqui é definido de acordo com o proposto no artigo de Grossman e Hart (1983). Naturalmente, ele depende de dois atores, principal e agente. Assume-se que após a formalização do contrato entre as duas partes, as ações do agente resultarão num valor y. Esse valor pertence a um conjunto Y tal que $y_1 < y_2 \cdots < y_n$, ou seja, Y detém todos os resultados possíveis de y. Evidentemente, esse resultado dependerá do esforço do agente descrito por L. Um componente aleatório também é levado em conta e é responsável por afetar os resultados do agente. Ambos atores estão cientes desse componente aleatório. Dessa maneira, como o resultado depende do esforço e de um componente aleatório, o resultado também será uma variável aleatória descrita da seguinte maneira:

$$P(y = y_i \mid L) = p_i(L), \text{ tal que } i \in \{1, 2, ..., n\}. \tag{2.1}$$

Dessa relação, pressupõe-se que $p_i(L) > 0$ para todo L. Isso implica que dado um nível L de esforço, qualquer resultado $y \in Y$ é possível, senão o principal poderia inferir o nível de esforço utilizado pelo agente por meio do resultado observado. Outro fator importante, é que se existe um conjunto de resultados $Y = \{y_1, \ \dots \ , y_n\}$ então a relação $\sum_{i=1}^{n} p_i(L) = 1$ deve ser verdadeira.

O principal, que é o contratante do agente, espera que o contrato seja cumprido pela segunda parte da melhor maneira possível de modo a receber os resultados de acordo com o estabelecido no contrato. Sua contrapartida é remunerar o agente pelo seu trabalho, seu salário,

w. Sua utilidade, U^P, depende do lucro, π, que é a diferença entre o resultado e o salário do agente.

$$U^P(y - w) = U^P(\pi). \tag{2.2}$$

Pressupõe-se que a função 2.2 é côncava, dessa maneira o principal é neutro ou avesso ao risco. Outra ideia importante é o fato de que a utilidade do principal não é modelada de maneira que seja diretamente relacionada ao esforço do agente, mas sim aos resultados que foram frutos de seu trabalho.

O próximo passo é modelar a utilidade do agente, U^A. Quanto maior for a remuneração pelo trabalho executado, maior será sua utilidade. Por outro lado, o esforço implica em um custo para ele que é deduzido de sua utilidade, dessa maneira:

$$U^A(w,\ L) = u(w) - d(L). \tag{2.3}$$

Na função 2.3, $d(L)$ descreve a desutilidade do esforço do agente e $u(w)$ a utilidade do salário. Assim como nas preferências do principal, supõe-se que o agente é neutro ou avesso ao risco. Dessa maneira, a utilidade marginal do salário é decrescente devido a concavidade da função que representa suas preferências. Por outro lado, pressupõe-se que a desutilidade marginal do esforço é crescente. Assume-se também que a função seja aditivamente separável com o intuito de simplificar a análise e a solução do problema, caso contrário a aversão ao risco do agente variaria de acordo com o esforço empregado na atividade. De acordo com Macho-Stadler e Pérez-Castrillo (1997), não há perda de generalidade ao se fazer essa simplificação. A mesma hipótese está presente no modelo de Grossman e Hart (1983).

As duas últimas funções descrevem o conflito de interesses entre o principal e o agente. De um lado o principal está interessado nos resultados disponibilizados pelo agente, enquanto o agente não se preocupa com eles. Por outro lado, o agente está interessado no esforço empregado em seu trabalho, mas o principal não. Por fim, há a ideia pertinente de que maiores esforços são responsáveis por melhores resultados.

O agente não tem poder de barganha com relação ao formato do contrato, cabendo a ele aceitar o que é proposto pelo principal ou rejeitar. A fim de modelar essa situação, estabelece-se que o agente possui uma relação contratual disponível a qualquer momento no mercado, denominada utilidade reserva, R, a qual ele utiliza como parâmetro para sua tomada de decisão.

Como mencionado, o esforço é variável relevante para o resultado do trabalho de um agente. No entanto, podem ocorrer dificuldades na hora de observá-lo ou verificá-lo, o que torna difícil para o principal estabelecer um contrato que identifique um nível ótimo de esforço. Portanto, o problema de risco moral surge nessa ocasião.

Um contrato cuja remuneração pelo esforço do agente é fixo, de maneira que não dependa dos resultados, fará com que o agente sempre empregue o menor esforço possível. A solução para tal problema, é a de que a remuneração do agente dependa dos resultados observados pelo principal. Dessa maneira, o principal busca criar um contrato que antecipe o comportamento do agente, maximizando seu lucro e, consequentemente, sua utilidade U^P. Para tal, duas restrições devem ser levadas em conta:

$$L \in \arg\max_{L} \Big\{ \sum_{i=1}^{n} p_i(L)\ u(w(y_i)) - d(L) \Big\} \tag{2.4}$$

e

$$\sum_{i=1}^{n} p_i(L)\ u(w(y_i)) - d(L) \geq \bar{U}. \tag{2.5}$$

A condição (2.4) é denominada restrição de incentivo. Ela busca modelar a última decisão do agente no cronograma mostrado na figura 2, o problema de risco moral, em que o agente decide o nível de esforço que irá empregar. Na condição (2.5), o agente decide se irá assinar o contrato comparando a utilidade recebida pelo contrato definido pelo principal e a solução de mercado. Ambas as condições se referem à utilidade esperada do agente, já que a natureza é responsável por determinar o estado do sistema.

O problema de maximização do principal e a sua solução analítica é a seguinte:

$$\begin{aligned} \max_{[L,\ \{w(y_i)\}_{i=1,\ \dots\ ,\ n}]} \quad & E\{U^P(\pi)\} \\ \text{s.t.} \quad & L \in \arg\max_{L}\ E\{U^A(w, L)\} \\ & E\{U^A(w, L)\} \geq R \end{aligned} \tag{2.6}$$

A formulação do problema parece simples, no entanto, sua solução não é trivial. Assumindo que a função de desutilidade do agente seja contínua, cada nível de esforço implicaria em uma taxa de salário ótima associada a ele. O processo de solução do problema envolveria a análise de múltiplas restrições enquanto uma delas já é por si só um problema de maximização, especificamente, a de incentivo, fazendo com que o problema todo seja definido como uma maximização dupla. Uma estratégia para lidar com a restrição de incentivo é tratá-la da seguinte maneira:

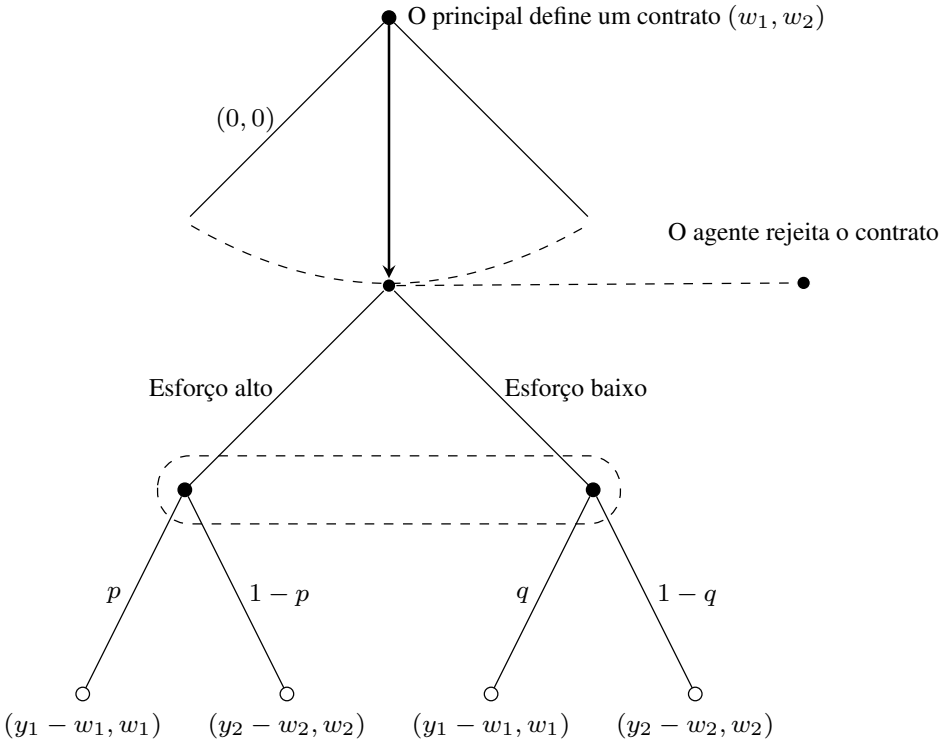
$$E\{U^A(w, \hat{L})\} \geq E\{U^A(w, L_j)\}, \quad j = 1, \dots,\ m.$$

No qual $\hat{L} \in \mathcal{L} = \{L_1,\ \dots\ ,\ L_m\}$ seria o esforço encarregado de fazer com que a utilidade esperada do agente atingisse seu maior valor.

Esse jogo pode ser modelado por meio de uma árvore de decisão que facilita a compreensão dos passos tomados pelo principal para solucionar o problema. A árvore leva em conta o cronograma definido na 2. Assume-se que existam dois níveis de esforço possíveis, alto e baixo. O principal leva em conta as probabilidades de sucesso condicionados aos níveis de esforço para definir as taxas de salário. Quando um alto nível de esforço é empregado, um resultado *bom* tem probabilidade p, enquanto um resultado *ruim* tem probabilidade $1 - p$. No caso de um esforço menor, a probabilidade de um resultado *bom* é q e de um resultado *ruim* é $1 - q$. O retângulo tracejado indica que o principal não consegue observar se o nível de esforço empregado foi alto ou baixo. Se as taxas definidas não respeitam a restrição de participação do agente, o jogo termina com a rejeição do contrato. A figura 5 descreve a situação.

É válido ressaltar que o salário não depende diretamente do esforço, mas sim da probabilidade de que um resultado bom esteja associado a um esforço específico. Dessa maneira, o contrato definido pelo principal é aquele que aumenta o seu lucro esperado, então, numa situação hipotética em que a probabilidade de obter o maior resultado esteja associada ao menor nível de esforço, o principal ofertará um contrato ao agente que o incentive a dedicar o menor nível de esforço possível. No capítulo de resultados, essa característica do modelo é analisada.

Figura 5 – Árvore de decisão do problema de Risco Moral com dois atores



Fonte: Elaborada pelo próprio autor.

2.3 O problema de seleção adversa

O problema de seleção adversa ocorre quando o agente detém informação privada relevante para o sucesso do contrato e não a revela para o principal. Um caso muito conhecido é a compra e venda de carros usados, estudada por Akerlof (1970). O mercado de veículos seminovos contém bens que vão de excelente a péssimo estado, mas saber tal informação com base no que se vê é extremamente difícil. Em algumas ocasiões, somente o dono do veículo detém informação relevante e pode não ser do interesse dele revelá-la, pois isso afetaria o preço do veículo ou a parte interessada poderia desistir de comprá-lo.

Essa situação não se restringe apenas ao mercado de carros, podendo ser utilizada para modelar os mais variados tipos de situações. Exemplos são relações de trabalho quando um principal tem de escolher entre vários tipos de agentes com comportamentos específicos que não são verificáveis antes da assinatura do contrato ou, como em Rothschild e Stiglitz (1976), para modelar o equilíbrio do mercado de apólices de seguro. No último caso, leva-se em conta que existem agentes que se arriscam muito ou pouco, mas esses não revelam informação a respeito de si mesmos.

Macho-Stadler e Pérez-Castrillo (1997) observa que a assimetria de informação pode fazer com que alguns mercados deixem de existir. Se, por exemplo, uma parte interessada em comprar um carro usado não consegue encontrar meios para verificar que está fazendo um bom negócio, pode ser melhor para ela simplesmente desistir da compra. A solução proposta para essa situação é *discriminar* a qualidade do produto através de contratos. Por exemplo, uma pessoa interessada em fazer uma pintura em sua casa tem a sua disposição dois pintores, um bom e outro ruim; o primeiro pintará a casa com a menor quantidade de tinta possível e fará um serviço limpo; o segundo fará um serviço sujo e que utilizaria maior quantidade de insumo; o principal sabendo dessas possibilidades, mas não podendo verificar em qual dos dois perfis os agentes se encaixam, pode determinar dois contratos que revelem informação dos atores.

Assim como no caso do modelo de risco-moral, assume-se que o resultado do principal depende do esforço empenhado pelo agente, entretanto, nesse caso o esforço é verificável. Assim sendo, descreve-se o lucro do principal como $\Pi(w, L) = \sum_{i=1}^{n} P_i(L)y_i - w$. Supõe-se também que a função de lucro do principal é côncava, implicando que ele seja neutro ou avesso ao risco.

Agora, diferente da situação anterior, leva-se em conta a existência de dois agentes que possuem diferenças com relação as suas preferências de utilizar esforço. O agente ruim possui maior desutilidade quando emprega uma quantidade de esforço L qualquer, enquanto o agente bom possui desutilidade menor. Esses agentes serão denominados respectivamente R e B. A constante k representa o efeito maior sobre a desutilidade do agente A.

$$\begin{aligned} U^R(w, L) &= u(w) - kd(L), \text{ com } k > 1 \\ U^B(w, L) &= u(w) - d(L). \end{aligned} \tag{2.7}$$

Com dois agentes, se o principal maximiza o lucro com relação ao esforço e ao salário enquanto utiliza como restrição $U^t \geq R$, em que $t = B, R$, um problema surge quando existe informação assimétrica. Assumindo uma quantidade L objetivada pelo principal, um contrato para U^B tem salário um pouco menor, porque a desutilidade é menor, mas o contrato para o agente U^R tem um salário um pouco maior. O agente ruim sempre aceitará o contrato que é voltado para ele, mas o agente bom está disposto a aceitar ambos contratos, o que não discrimina os agentes de maneira eficiente.

Nesse caso, é preciso levar em conta a proporção de agentes bons e ruins num mercado hipotético. Assume-se que a probabilidade de encontrar um agente do tipo ruim seja $0 < \pi^R < 1$. Sabendo disso, o principal pode construir um cardápio de contratos levando em conta as preferências dos agentes. Dessa maneira, se o agente t escolhe o contrato (L^t, w^t), ele está maximizando sua utilidade e o principal o seu lucro esperado. É importante então que esses contratos sejam auto-exequíveis: os agentes escolhem o que é melhor para eles. Por fim, se o número de agentes é n, o número de contratos será n e não $n + 1$, caso contrário sempre sobraria um contrato.

Levando em conta as observações anteriores, o problema de maximização do lucro esperado do principal será dado por:

$$\begin{aligned} \max_{[(w^B,L^B),(w^R,L^R)]} \quad & \theta^R\big[\Pi(w^R, L^R)\big] + (1-\theta^R)\big[\Pi(w^B, L^B)\big] \\ \text{s.t.} \quad & u(w^B) - v(L^B) \geq R \\ & u(w^R) - kv(L^R) \geq R \\ & u(w^B) - v(L^B) \geq u(w^R) - v(L^R) \\ & u(w^R) - kv(L^R) \geq u(w^B) - kv(L^B) \end{aligned} \tag{2.8}$$

As duas primeiras restrições são denominadas restrições de participação e as duas últimas são de compatibilidade de incentivo. Os contratos criados precisam proporcionar uma utilidade maior ou igual a outras oportunidades disponíveis no mercado. Além disso, o contrato do agente B proporciona para ele uma utilidade maior ou igual a utilidade proporcionada pelo contrato voltado ao agente R. A mesma coisa deve ocorrer no caso do agente R, ou seja, o seu contrato é a melhor escolha possível. Das três primeiras restrições, nota-se que o principal só precisa se preocupar com a restrição de participação do agente menos eficiente com relação ao esforço, porque:

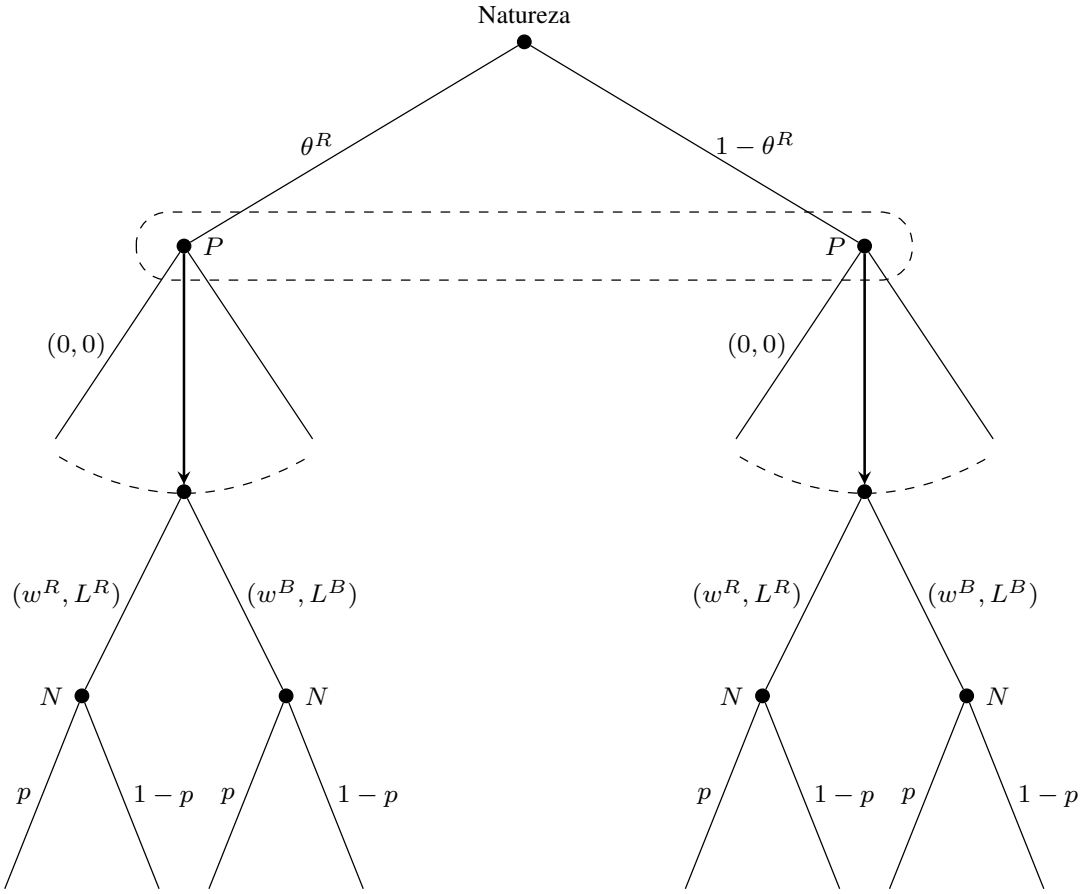
$$u(w^B) - v(L^B) \geq u(w^R) - v(L^R) \geq u(w^R) - kv(L^R) \geq R$$

Uma condição extra importante que surge das duas últimas restrições implica que o maior nível de esforço deve ser demandado do agente mais eficiente ($L^B \geq L^R$), pois:

$$v(L^G) - v(L^R) \leq u(w^G) - u(w^R) \leq k[v(e^G) - v(e^B)].$$

Com base na figura 3, uma árvore de decisão foi construída . A natureza decide qual tipo de agente (R ou B) por meio da proporção θ^R. O agente escolhe um contrato que o agrade, mas o principal não consegue observar o tipo de agente, característica descrita pelos retângulos pontilhados. Os contratos são compostos por uma remuneração específica e o nível de esforço de acordo com cada tipo de agente (w^t, L^t), em que o subscrito t denota o tipo de agente. A depender do contrato, diferentes quantidades de esforço levam a diferentes resultados (omitidos para fins de simplificação). Assume-se que o empenho do contrato escolhido tenha sido empregado pelo agente contratado (independente do seu tipo) e, além disso, dentro desse jogo, o empenho que compõe os contratos definidos pelo principal traz o melhor resultado possível a uma probabilidade p, ou seja, o sucesso é definido pela natureza.

Figura 6 – Árvore de decisão do problema de Seleção Adversa com dois atores



Fonte: Elaborada pelo próprio autor.

3 Revisão de métodos numéricos

Neste capítulo, todas as técnicas empregadas para solucionar os problemas definidos no capítulo anterior serão apresentados e discutidos. Os problemas tratados estão presentes em Judd (1998, p. 128). A demonstração da aplicação dos métodos de penalidade e as funções que passaram pela aplicação estarão presentes no apêndice em conjunto com os códigos de programação para a solução do problema numericamente.

3.1 Função Penalidade

Em problemas de otimização, a adição de uma restrição extra torna a solução do problema ligeiramente mais difícil. Uma alternativa é transformar um problema de otimização condicionada num problema não-condicionado. Para isto, basta alterar a função-objetivo de maneira que uma violação nas restrições altere a função. Considere um problema de duas restrições, como em Judd (1998):

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t} \quad & g(x) = a, \\ & h(x) \leq b. \end{aligned}$$

Como esse é um problema de minimização, a função penalidade ficaria da seguinte maneira:

$$\min_{x} f(x) + \frac{1}{2} P \cdot \left(\sum_{i} (g^i(x) - a_i)^2 + \sum_{j} (\max[0, h^j(x) - b_j])^2 \right)$$

Toda vez que uma das restrições é violada, o valor da restrição é adicionado. No caso de um problema de maximização, os valores referentes a violações são subtraídos. Por si só essa função não é capaz de solucionar o problema em si e, por isso, precisa ser aliada a outros métodos que ajudem a encontrar seus pontos de otimização.

Alguns problemas podem ocorrer quando a função de penalidade é criada, como distorções nos valores da função e de sua inclinação, impossibilitando a diferenciação em alguns pontos. Isso é relevante porque existem algoritmos que encontram a solução para esse tipo problema podem depender de derivadas numéricas. O parâmetro adicionado P tem como objetivo aliviar esses possíveis problemas. Tipicamente, o parâmetro é definido como 10^k, onde k é o número da tentativa de minimização. A cada nova iteração o valor de k é modificado até o ponto que os resultados convirjam.

3.2 O método de Newton

Para encontrar soluções de problemas de funções penalidade através de métodos computacionais, alguns algoritmos iterativos podem ser aplicados. Algoritmos são conjuntos de ações finitas que visam resolver algum tipo de problema. Os algoritmos iterativos dependem de um valor inicial que é utilizado para estimar o valor que soluciona o problema. O método de Newton é um desses instrumentos.

Judd (1998) aponta a existência de múltiplas maneiras quando o assunto é implementar o método de Newton para múltiplas variáveis. Para resolver o problema em questão, buscou-se implementar uma versão genérica do método que utiliza derivadas para computar soluções de otimização.

Seja f: $\mathbb{R}^n \to \mathbb{R}$ uma função qualquer, seu vetor gradiente, $\nabla f(x)$, e sua matriz Hessiana, $H(x)$ serão dados, respectivamente, pelas primeiras derivadas de cada variável presente em $\mathbf{x}$ e as derivadas parciais de segunda ordem:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right), \; H(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1^2}(x) & \ldots & \frac{\partial f}{\partial x_1 x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n x_1}(x) & \ldots & \frac{\partial f}{\partial x_n^2} \end{bmatrix}$$

O método de Newton utiliza aproximações locais quadráticas da função f para estimar um ponto crítico x^* e mínimo local. Isso permite que funções convexas, mas não quadráticas possam ser avaliadas pelo método, por exemplo, a função de utilidade de um principal neutro ao risco. Um "chute"inicial é utilizado para avaliar a função. Então, $f(x)$ é substituída por uma aproximação de f em x^k dada por:

$$f(x) \doteq f(x^k) + \nabla f(x^k)(x - x^k) + \frac{1}{2}(x - x^k)^\top H(x^k)(x - x^k)$$

Se f for uma aproximação convexa, então $H(x^k)$ será positiva definida; a aproximação terá um mínimo $x^k - H(x^k)^{-1}\nabla(f(x^k))^\top$, o qual será denominado x^{k+1}. Esse valor será utilizado na próxima iteração, ou seja: uma nova aproximação local de f em x^{k+1} será encontrada e minimizada até que o algoritmo atinja um determinado objetivo. Esse passo, no entanto, pode ser readaptado para que o computador não precise calcular a inversa da matriz de Hessian. Basta notar que $s \overset{\equiv}{} x^{k+1} - x^k$. A partir desse passo, basta resolver o sistema linear:

$$H(x^k)s^k = -(\nabla f(x^k))^\top$$

Para que o algoritmo não continue calculando valores de maneira indefinida, dois testes são feitos com base no vetor gradiente de $f(x^k)$, o valor de $f(x^k)$, o próprio x^k e a hessiana de $f(x^k)$. O primeiro teste verifica se os valores encontrados x^k e x^{k+1} convergiram:

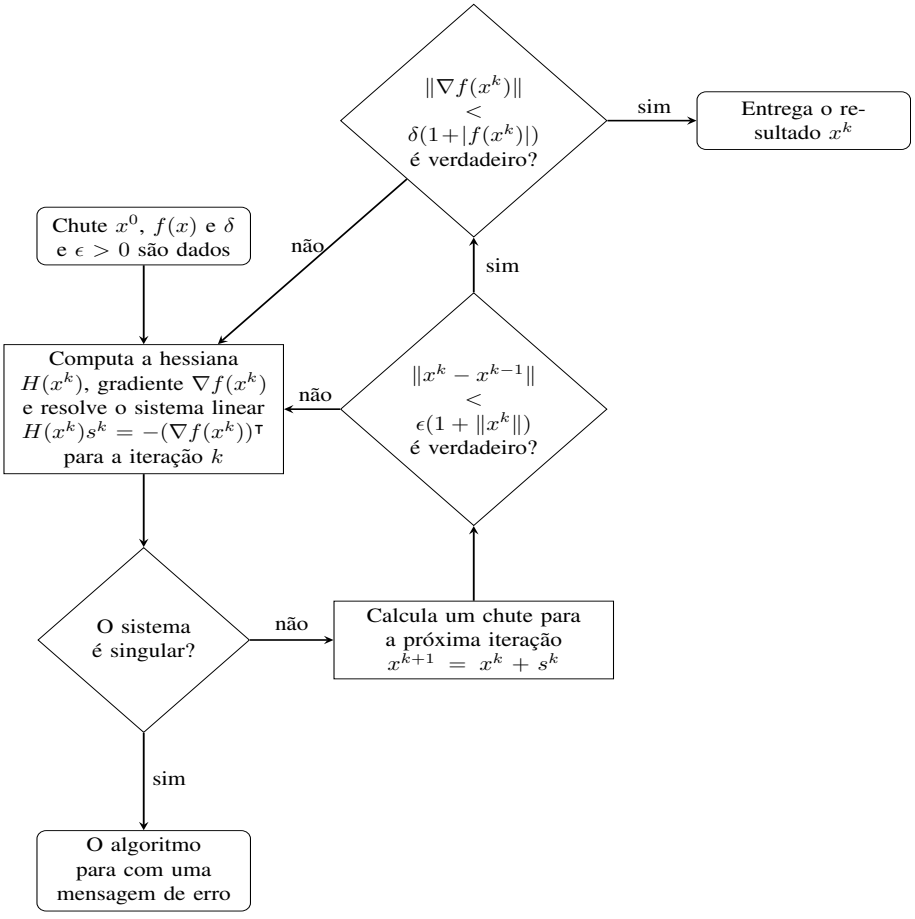
$$\|x^{k+1} - x^k\| < \epsilon(1 + \|x^k\|)$$

Como x^k e x^{k+1} são vetores, a norma é utilizada para verificar a diferença entre seus tamanhos. ϵ é um parâmetro positivo e arbitrariamente próximo de zero cujo objetivo é definir a precisão com que os valores sejam calculados. No caso da assertiva ser verdadeira, o segundo teste verifica se o último valor encontrado de x^k satisfaz a condição de primeira ordem, verificando se $\nabla f(x^k)$ é zero ou arbitrariamente próximo de zero:

$$\|\nabla f(x^k)\| \leq \delta(1 + |f(x^k)|)$$

O valor 1 é adicionado a $f(x^k)$ de modo a garantir que os casos em que $f(x^k) \approx 0$ sejam avaliados corretamente. δ possui a mesma função que ϵ.

Figura 7 – Fluxograma método de Newton



Fonte: Elaborada pelo próprio autor.

Alguns problemas podem ocorrer com o algoritmo. Como a sequência de passos depende de derivadas numéricas, se a função não é diferenciável, é bastante provável que a solução

encontrada não seja verdadeira ou que o método simplesmente não funcione. Além disso, o método utiliza condições de primeira ordem para encontrar pontos ótimos, o que não garante que o ponto encontrado seja um máximo ou mínimo global. Outro possível problema é a singularidade de matrizes: como o método depende da solução de um sistema linear, existem ocasiões em que a matriz hessiana numérica seja singular ou fique distorcida pelos altos valores do parâmetro P escolhidos para a função penalidade.

3.2.1 Implementação

```
1
2     chute <- c(0,0)
3     n <- 1000
4     xk <- matrix(chute, byrow = TRUE)
5     k <- list(xk)
6
7     for(i in 1:n){
8     hessiana <- rootSolve::hessian(f = func, x = chute, centered = TRUE)
9     gradiente.transposta <- -1 * t(rootSolve::gradient(f = PenalFunct,
10    x = chute,
11    centered = TRUE))
12
13    sk <- qr.solve(hessiana, gradiente.transposta)
14    xk.1 <- xk + sk
15    k[[i]] <- c(xk.1)
```

O algoritmo inicia com a definição de um "chute"da região onde a solução do problema se encontra. Nesse caso, o algoritmo é implementado para uma função de duas variáveis e, para tanto, o chute possui dois valores. O objeto "n"descreve o número de iterações ou tentativas que o programa irá executar o código do algoritmo em questão. A matriz "xk"é um objeto criado para que as operações do algoritmo sejam efetuadas, por fim, uma lista "k"é criada e nela toda as aproximações encontradas pelo método serão armazenadas nela.

A partir da linha 5, uma iteração é iniciada e ela ocorrerá a depender do valor do parâmetro "n". Por exemplo, se $n = 10$, dez tentativas serão efetuadas. A cada passo, como observado nas linhas 6, 7, 8 e 9, a matriz hessiana e vetor gradiente de uma função qualquer é avaliada no ponto definido por "chute", o argumento *centered* = *TRUE* garante que as derivadas numéricas utilizarão o método de diferença centrada.

Na linha 11, o sistema linear entre a hessiana e gradiente.transposta é resolvido. Na linha 12, os resultados encontrados a partir do sistema são adicionados ao chute definido inicialmente. Esse novo resultado, um vetor, é adicionado como sendo o i-ésimo item da lista "k".

```
14    first.test <- (norm(xk - xk.1) < epsilon * (1 + norm(xk)))
15    g <- first.test
16
17    if(first.test == TRUE){
18    second.test <- (norm(gradiente.transposta) <= delta * (1 + abs(func(xk))))
19    h <- second.test
```

```
20      if(second.test == TRUE){
21      root.approx <- unlist(tail(k, n = 1))
22      names(root.approx) <- c('x', 'y')
23      print(root.approx)
24      }
25      }
26      xk <- xk.1
27      chute <- c(xk.1)
28      }
```

Após os cálculos terem sido feitos, é preciso avaliar se a solução "xk.1"encontrada é a correta. A linha 14 avalia o primeiro teste, criando um objeto cujo valor é verdadeiro ou falso. O teste verifica se a diferença entre o chute e o resultado encontrado é menor que um parâmetro de erro. Se o primeiro teste for verdadeiro, fato avaliado pela linha 17, o código das linhas 18 a 20 é executado. Um segundo teste que verifica se a norma da gradiente transposta é menor que o valor absoluto da função analisada no ponto do chute é executado e, caso a resposta seja verdadeira, as linhas 21, 22 e 23 são executadas. Nessas linhas, o último valor adicionado a lista k é utilizado para criar um vetor e seus valores são nomeados x e y e depois são mostrados no console do programa. Para o caso do segundo teste não ser verdadeiro, o último resultado encontrado se torna o novo chute e o código é reexecutado.

3.3 O método de diferença finita

Os resultados das hessianas e gradientes serão calculados através de aproximações numéricas utilizando o método de diferença finita. Especificamente, para encontrar derivadas, a diferença centrada será utilizada para calcular esses valores garantindo maior precisão de resultados. Supondo $f\colon \mathbb{R} \to \mathbb{R}$, sua derivada será dada por:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

A diferença centrada de uma função é dada por:

$$f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

Dividindo a diferença centrada por h, definido por $max\{\epsilon|x|, |x|\}$, sendo ϵ um valor pequeno tipicamente inferior a 10^{-6}, pode-se encontrar uma aproximação numérica da derivada:

$$f'(x) \approx \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}$$

Esse método pode ser utilizado para encontrar o vetor gradiente de uma função qualquer. Cada elemento da matriz hessiana será calculado levando em conta o fato de que a segunda derivada numérica seria a diferença centrada entre as primeiras derivadas da função, assim sendo:

$$\begin{aligned} f''(x) \approx & \frac{\frac{f(x+\frac{1}{2}h+\frac{1}{2}h)-f(x-\frac{1}{2}h+\frac{1}{2}h)}{h} - \frac{f(x+\frac{1}{2}h-\frac{1}{2}h)-f(x-\frac{1}{2}h-\frac{1}{2}h)}{h}}{h} \\ = & \frac{f(x+h)-f(x)-f(x)+f(x-h)}{h^2} \\ = & \frac{f(x+h)-2f(x)+f(x-h)}{h^2} \end{aligned}$$

Genericamente, supondo $f\colon \mathbb{R}^n \to \mathbb{R}$, a primeira derivada do i-ésimo elemento é dada por:

$$\frac{\partial f}{\partial x_i} \doteq \frac{f(x_1, \ldots, x_i + \frac{h_i}{2}, \ldots, x_n) - f(x_1, \ldots, x_i - \frac{h_i}{2}, \ldots, x_n)}{h_i}$$

Algum grau de erro com relação às soluções analíticas pode surgir em decorrência da utilização desses métodos. No entanto, segundo Judd (1998 apud CARTER, 1993), algoritmos de otimização podem gerar bons resultados mesmo quando as hessianas e gradientes numéricas possuem erros de até 50% quando comparados com suas soluções analíticas, o que garante certo grau de confiabilidade quanto a eficácia do método. Uma alternativa para calcular derivadas analíticas computacionalmente e, consequentemente, minimizar erros, é a utilização de computação simbólica, mas o método está fora do escopo desse trabalho.

3.4 O método do polítopo

Nelder e Mead (1965) propuzeram uma alternativa para minimizar (maximizar) funções com múltiplas variáveis. Por exemplo, supondo $f\colon \mathbb{R}^n \to \mathbb{R}$, o seu método implicaria na avaliação dessa mesma função em $n+1$ vértices de um objeto contido em $\mathbb{R}^n$, denominado polítopo. O polítopo, então, teria seu formato modificado através da transformação dos vértices a depender dos valores encontrados até o momento em que os vértices convirjam a um ponto de máximo ou mínimo.

O processo de transformação depende de uma sequência de passos que são repetidos iterativamente. No trabalho original, os autores utilizam três transformações possíveis que dependem do valor da função em cada um dos $n+1$ vértices: reflexão, contração e expansão. Nesse trabalho, o método utiliza uma transformação adicional: o encolhimento. Isso amplia o número de ferramentas possíveis para a transformação do polítopo e permite que um ponto ótimo seja encontrado de maneira mais robusta.

Para o caso de uma minimização, algoritmo sucederia da seguinte maneira: assumindo $f\colon \mathbb{R}^n \to \mathbb{R}$, os vértices de teste seriam $\boldsymbol{X_1}, \ldots, \boldsymbol{X_{n+1}}$. No primeiro passo, a função seria avaliada em cada um dos pontos e seus valores seriam ordenados da seguinte maneira:

$$f(\boldsymbol{X_1}) \leq f(\boldsymbol{X_2}) \leq \ldots \leq f(\boldsymbol{X_{n+1}})$$

Então, a reflexão desse polítopo é calculada. Para encontrá-la é preciso calcular o centroide de todos os pontos, com exceção de $\boldsymbol{X_{n+1}}$:

$$\boldsymbol{X_o} = \frac{\boldsymbol{X_1} + \boldsymbol{X_2} + \ldots + \boldsymbol{X_n}}{n}$$

Na equação anterior, o subscrito $_o$ indica que a variável se refere ao centroide. O terceiro passo, então, utilizando $\boldsymbol{X_{n+1}}$ como referência, é calcular a reflexão que será dada por:

$$\boldsymbol{X_r} = \boldsymbol{X_o} + \alpha(\boldsymbol{X_o} - \boldsymbol{X_{n+1}}) \tag{3.1}$$

A função é avaliada no ponto de reflexão ($\boldsymbol{X_r}$) do polítopo e é comparada com os valores anteriormente obtidos. Se $f(\boldsymbol{X_r})$ for melhor que o segundo melhor valor, mas não melhor que o primeiro valor, o ponto $\boldsymbol{X_{n+1}}$ é substituído por $\boldsymbol{X_r}$ e o algoritmo volta ao primeiro passo.

O quarto passo só é levado em conta se $f(\boldsymbol{X_r})$ for o melhor resultado até então ($f(\boldsymbol{X_r}) > f(\boldsymbol{X_1})$). Uma expansão do polítopo é executada, tomando como referência o vértice de reflexão, dessa maneira:

$$\boldsymbol{X_e} = \boldsymbol{X_o} + \gamma(\boldsymbol{X_r} - \boldsymbol{X_o}) \tag{3.2}$$

Se $f(\boldsymbol{X_e}) \leq f(\boldsymbol{X_r})$, então o algoritmo volta ao primeiro passo substituindo $\boldsymbol{X_{n+1}}$ por $\boldsymbol{X_e}$. Caso contrário, $\boldsymbol{X_{n+1}}$ é substituído por $\boldsymbol{X_r}$.

Os próximos passos envolvem contrações e encolhimento do polítopo. O quinto passo só é executado se $f(\boldsymbol{X_1}) \leq \ldots \leq f(\boldsymbol{X_r}) \leq f(\boldsymbol{X_{n+1}})$, ou seja, se o vértice de reflexão só for melhor que o pior vértice do polítopo. Nesse caso, a contração é calculada tendo como referência o vértice de reflexão:

$$\boldsymbol{X_c} = \boldsymbol{X_o} + \rho(\boldsymbol{X_r} - \boldsymbol{X_o}) \tag{3.3}$$

Se $f(\boldsymbol{X_c}) \leq f(\boldsymbol{X_r})$, o vértice $\boldsymbol{X_{n+1}}$ é substituído pelo vértice contraído. Por outro lado, se esse não for o caso, o polítopo será encolhido. Todos os pontos serão substituídos utilizando a fórmula de encolhimento, com exceção do melhor ponto definido por $\boldsymbol{X_1}$:

$$\boldsymbol{X_i} = \boldsymbol{X_1} + \sigma(\boldsymbol{X_i} - \boldsymbol{X_1}) \tag{3.4}$$

Por fim, caso o vértice referente a reflexão seja de fato o pior dos vértices, uma contração do polítopo inicialmente provido é executada. Se o ponto de contração ($f(\boldsymbol{X_c})$) for menor ou igual ao pior vértice ($f(\boldsymbol{X_{n+1}})$), uma substituição dos pontos ocorre. Caso contrário, o polítopo será encolhido. De acordo com Nash (1990, p. 170), os parâmetros definidos nas equações 3.1, 3.2, 3.3 e 3.4 não possuem regra ou critério, sendo definidos heuristicamente. Os valores

usados no algoritmo são $\alpha = 1$, $\lambda = 2, \rho = \sigma = 0.5$. Os pontos iniciais para a execução do programa também devem ser escolhidos heuristicamente, mas com cautela, caso contrário a solução encontrada pode divergir muito da real.

Para o caso da contração, existem ocasiões em que o polítopo de fato não é contraído ou encolhido, fazendo com que a continuidade do algoritmo seja em vão ou não gere resultados adequados. Nash (1990) sugere que, para garantir maior robustez no passo de contração, toda vez que o passo for executado e $f(\boldsymbol{X_c}) \leq f(\boldsymbol{X_r})$, o tamanho do novo polítopo, mensurado pela soma da norma dos vértices, será comparado com o anterior através da seguinte equação:

$$\sum_{i=1}^{n+1} \|\boldsymbol{X_i^c} - \boldsymbol{X_{n+1}^c}\| - \sum_{i=1}^{n+1} \|\boldsymbol{X_i} - \boldsymbol{X_{n+1}}\| \tag{3.5}$$

O sobrescrito c denota que o polítopo em questão contém o vértice de contração. Se a diferença entre os tamanhos, dados pelas normas calculadas, for muito próxima de zero, o algoritmo pula para o primeiro passo sem incluir o vértice de contração ou encolhe o polítopo ainda mais, a depender do passo em questão.

Posto isso, é preciso determinar a condição de parada do algoritmo ou ele continuará os passos para sempre. A condição proposta por Nelder e Mead (1965) é a utilização do desvio-padrão dos valores da função em cada vértice: se o desvio-padrão dos resultados for menor ou igual a um certo critério ϵ, o código para e a solução é entregue. Neste trabalho, o critério empregado foi este. No entanto, é preciso ressaltar um possível problema que pode ocorrer ao escolher esse critério, como a terminação prematura do programa em funções que tenham superfícies planas. Press et al. (2007, p. 290) sugerem que uma vez que o programa é terminado, é preciso repetí-lo com a solução encontrada de modo a diminuir a possibilidade de erro. Outro critério interessante para a solução do problema é a utilização da distância euclideana dos vértices.

Diferente do método de Newton, esse método não depende de condições de primeira ordem para encontrar um ponto mínimo, se mostrando como uma alternativa para problemas em que o primeiro método falha (por exemplo, singularidade). Ambos devem ser usados de maneira complementar, no entanto, como sugere Judd (1998, p. 143). Dependendo do problema, como no caso de funções com superfícies lisas, o método de Newton deve ser empregado. Para o caso de funções cujas superfícies sejam rugosas (muitos mínimos e máximos localizados na região de solução do problema), o método do polítopo, aliado a várias reinicializações para garantir que o ponto encontrado não seja uma solução local, se mostra como o método mais adequado.

3.4.1 Implementação

Como cada polítopo depende de um certo número de variáveis que representam seus vértices, decidiu-se implementar o algoritmo de maneira que um polítopo seja uma matriz de

dados, assim sendo, as colunas são variáveis e linhas são seus os respectivos vértices. Para tanto, o pacote de funções *tidyverse*, disponibilizado por Wickham (2017), foi utilizado. Seu pacote permite a manipulação de matrizes de dados de maneira simples e intuitiva. A seguir, o primeiro bloco de código é implementado e analisado.

```
1    library(tidyverse)
2
3    calculate_size <- function(data){
4    size <- 0
5    for(i in 1:nrow(data)){
6    vector_size <- data[i, ] - data[nrow(data),]
7    size <- size + norm(vector_size, type = '2')
8    }
9    return(size)
10   }
11
12   evaluate_function <- function(list, func){
13   std <- reduce(list, bind_rows) %>%
14   mutate(values = pmap_dbl(., func)) %>%
15   .$values %>%
16   sd()
17   return(std)
18   }
```

A primeira linha ativa o pacote de *tidyverse*. As linhas 3 a 10 implementam uma função que verifica o tamanho de um polítopo. A segunda função, nas linhas 12 a 18, calcula o desvio padrão das funções-objetivo nos vértices do polítopo. Essas são funções necessárias para o processo de implementação do algoritmo e, por isso, são iniciadas no começo dele.

```
1    init_mat <- reduce(vertices, bind_rows) %>%
2    mutate(values = pmap_dbl(., func)) %>%
3    arrange(desc(values)) %>%
4    mutate(id = row_number())
5
6    if(sd(init_mat$values, na.rm = TRUE) <= 0){
7    return(init_mat[, 1:(ncol(init_mat)-2)])
8    }else{
9    centroid <- colSums(init_mat[1:(nrow(init_mat)-1), 1:(ncol(init_mat)-2)]) *
10   (1/(nrow(init_mat)-1))
11   reflection <- c(centroid + (centroid - c(filter(init_mat, id == max(id)),
12   recursive = TRUE)[1:(ncol(init_mat)-2)]), id = nrow(init_mat) + 1)
```

As primeiras quatro linhas transformam uma lista de vetores que contenham os vértices iniciais a serem avaliados pelo algoritmo numa matriz de dados. A função-objetivo é avaliada nos vértices e seus resultados são organizados do maior para o menor valor e identificados por um "id". O "id"auxilia na comparação antes e após novas operações.

Das linhas 6 a 8, o desvio padrão das funções-objetivo nos vértices são avaliados e se forem menores ou iguais a zero, a primeira linha da matriz de vértices é retornada como resultado. Caso contrário, o processo de reflexão do polítopo é iniciado, primeiro com o cálculo do centroide e, depois, como cálculo da reflexão.

```
13    evaluate <- bind_rows(init_mat, reflection) %>%
14    mutate(values = pmap_dbl(.[, 1:(ncol(init_mat)-2)], func)) %>%
15    arrange(desc(values)) %>%
16    select(-values)
17
18    if(between(which(evaluate$id == max(evaluate$id)), 2, (max(evaluate$id) - 2))){
19    return(transpose(select(evaluate, -id)[1:(nrow(init_mat)), ]))
```

Nas linhas 13 a 16, a reflexão é agregada como uma linha à matriz de vértices e seu "id"é igual ao número de vértices iniciais mais uma unidade. O valor da função objetivo nesse no vértice é calculado e o a matriz é reordenada do menor para o maior valor. A linha 18 verifica se o índice do vértice de reflexão não é o menor, o que indicaria que esse é o vértice mais próximo da solução buscada, nem está entre os piores. Se a condição for verdadeira, a matriz de dados é retornada como uma lista de $n + 1$ elementos, onde n é o número de variáveis da função objetivo. O formato de lista é relevante pois esse passo não conclui encontra a solução de fato, precisando ser reavaliada.

```
20    }else if(which(evaluate$id == max(evaluate$id)) == 1){
21
22    expansion <- c(centroid + 2 * (c(filter(evaluate, id == max(id)),
23    recursive = TRUE)[1:(ncol(evaluate)-1)] -
24    centroid), id = nrow(evaluate)+1)
25
26    evaluate_test <- bind_rows(evaluate, expansion) %>%
27    mutate(values = pmap_dbl(.[, 1:(ncol(evaluate)-1)], func)) %>%
28    arrange(desc(values))
29
30    if(which(evaluate_test$id == max(evaluate_test$id)) < which(evaluate_test$id == (max(
          evaluate_test$id)-1))){
31    return(transpose(select(evaluate_test, -(id:values))[1:(nrow(evaluate_test)-2), ]))
32    }else{
33    return(transpose(select(evaluate, -(id))[1:(nrow(evaluate)-1), ]))
34    }
```

Caso o último bloco não tenha funcionado, os blocos 20 a 31 são executados. Nesse passo, o algoritmo verifica se o vértice de reflexão possui o menor índice possível. Se verdadeiro, o passo de expansão é iniciado. O vértice de reflexão é multiplicado por uma constante (2) e somado ao centroide e depois seu valor é avaliado na função-objetivo e reordenado. Nesse ponto, um teste verifica se o ponto de expansão gera resultados melhores que o ponto de reflexão. Se verdadeiro, a matriz é retornada como uma lista, para que esses vértices sejam reavaliados pelo algoritmo. Caso contrário, o vértice de expansão é removido e o que sobra é retornado como uma lista para reavaliação.

```
35    else if(which(evaluate$id == max(evaluate$id)) == (max(evaluate$id) - 1)){
36    contraction <- c(centroid + 0.5*(c(filter(evaluate, id == (max(id)-1)),
37    recursive =TRUE)[1:(ncol(evaluate)-1)] - centroid),
38    id = nrow(evaluate) + 1)
39
```

```
40      evaluate <- bind_rows(evaluate, contraction) %>%
41      mutate(values = pmap_dbl(.[, 1:(ncol(evaluate)-1)], func)) %>%
42      arrange(desc(values)) %>%
43      select(-values)
44
45      contraction_test <- near(calculate_size(select(evaluate, -id)[1:nrow(init_mat),]) -
46      calculate_size(select(init_mat, -(id:values))[1:nrow(init_mat),]), 0)
```

Nesse bloco, o algoritmo definiu que o índice do vértice de reflexão não era o melhor possível. Desse modo, ele verifica se o índice do vértice de reflexão é igual ao id do último vértice antes da criação da reflexão. Caso seja verdadeiro, o processo de contração é iniciado nas linhas 36 a 43. O vértice de contração é calculado, avaliado na função-objetivo e reordenado. Por fim, nas linhas 45 a 46, compara-se a diferença entre o polítopo sem o vértice de contração com o que possui o novo vértice e testa-se se esse valor é próximo de zero.

```
47      if(which(evaluate$id == (max(evaluate$id)-2)) > which(evaluate$id == max(evaluate$id)) &
48      contraction_test != TRUE){
49      return(transpose(select(evaluate, -id)[1:(nrow(init_mat)), ]))
50      }else{
51      best_vec <- c(select(evaluate, -id)[1, ], recursive = TRUE)
52      best <- evaluate[1, 1:(ncol(evaluate)-1)]
53      for(i in 1:(nrow(evaluate)-3)){
54      shrink <- best_vec + 0.5*(c(evaluate[i+1, 1:(ncol(evaluate)-1)], recursive = TRUE) - best_
            vec)
55      best <- bind_rows(best, shrink)
56      }
57      return(transpose(best))
```

As linhas 47 a 49 determinam que caso o índice do vértice de expansão seja menor que o índice do vértice de reflexão e o tamanho do novo polítopo seja diferente do polítopo anterior, a matriz é retornada como uma lista para que uma reavaliação seja realizada. Caso contrário, nas linhas 51 a 57, o processo de encolhimento do polítopo é iniciado e depois retornado como uma lista para reavaliação. É importante observar que no processo de encolhimento, cada vértice, com exceção do primeiro, é encolhido.

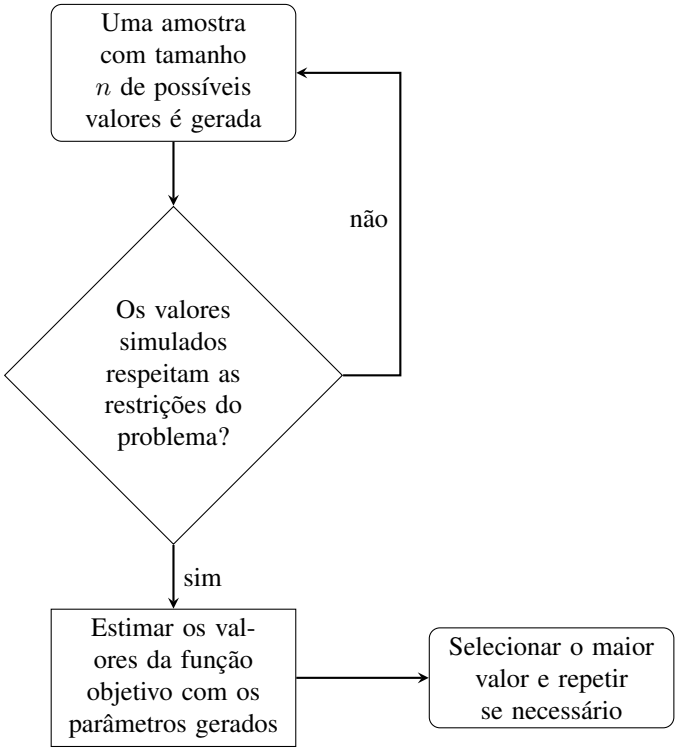
Para o caso de nenhuma das opções anteriores serem verdadeiras, novamente, a contração do polítopo é verificada e, caso não traga uma solução ideal, seu encolhimento é realizado. Para a implementação completa, verificar o apêndice A.

3.5 O método de força bruta

Levando em conta uma função qualquer $f\colon \mathbb{R}^n \to \mathbb{R}$ e $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, é possível fazer um estudo de como $f(\boldsymbol{x})$ se comporta a medida que $\boldsymbol{x}$ varia. De acordo com Linge e Langtangen (2016), tal método pode ser utilizado para fins de otimização ou até mesmo encontrar raízes.

Toda solução encontrada para os problemas analiticamente definidos nesse trabalho deve respeitar as restrições definidas. Para encontrar possíveis valores que solucionem os problemas, então, basta definir um algoritmo simples que simule uma amostra grande de soluções candidatas para o problema. Daí, basta verificar quais soluções geradas respeitam as restrições e, por fim, testar quais soluções geram os maiores (menores) valores possíveis. A figura explica o processo esquematicamente.

Figura 8 – Fluxograma método de força bruta



Fonte: Elaborada pelo próprio autor.

A verificação das soluções encontradas através dos métodos anteriores é outro possível emprego para o método. Pode-se estimar se os valores esperados encontrados através dos outros métodos são equiparáveis com uma simulação de amostra grande usando a média cumulativa para cada novo passo. Outra aplicação possível é o estudo da sensibilidade do valor da função objetivo ao se alterar seus parâmetros, por exemplo, a alteração das probabilidades de sucesso quando um agente desempenha o maior esforço possível. É válido observar, no entanto, que devido ao grande número de pontos gerados que precisam ser alocados na memória de um computador e depois calculados, esse pode ser um método bastante lento para se encontrar soluções.

4 Resultados

Neste capítulo os resultados obtidos ao aplicar os métodos mencionados no capítulo de Metodologia serão discutidos.

4.1 Soluções para o problema de risco moral

O método de Newton foi aplicado na função de penalidade obtida através do problema mencionado por Judd (1998, p. 129) ao passo em que se aumentava a taxa de penalidade de 10 a 1.000.000. Conforme a penalidade variava positivamente, novas taxas de salário para cada resultado foram obtidas. Para verificar se os salários calculados eram de fato corretos, as violações das restrições de incentivo (VI) e a violação do contrato oferecido com relação a melhor opção do mercado (VR) foram calculadas. Para ambos os casos, quanto mais próximo de zero melhor.

Tabela 1 – Resultados da aplicação do método de Newton no problema de Risco Moral

w_1	w_2	P	VI	VR
0,788879	0,460472	10	-0,023318	-0,109443
1,110971	0,535550	100	-0,006307	-0,019910
1,214240	0,545204	1.000	-0,000837	-0,002417
1,228876	0,546201	10.000	-0,000087	-0,000248
1,230412	0,546301	100.000	-0,000009	-0,000025
1,230566	0,546311	1.000.000	-0,000001	-0,000002

Fonte: Elaborada pelo próprio autor.

Os resultados foram bastante similares aos obtidos em Judd (1998). Quando $y = 2$, o agente será remunerado em \$1,23. Por outro lado, o agente será remunerado em apenas \$0,55 quando o resultado for $y = 0$. Levando em conta a distribuição das probabilidades no caso em que o maior nível de esforço é empregado, já que esse é o esforço visado pelo principal, o salário esperado do agente será de \$1,0937, superando a opção disponível de \$1,00 em \$0,0937. Esse salário maior compensa o risco que o agente lida ao aceitar essa proposta.

Quando outros algoritmos são aplicados ao problema, os salários convergem para o mesmo valor encontrado pelo método de Newton. No entanto, o tempo de execução dos algoritmos varia bastante. Nos testes feitos, o algoritmo de Newton levou cerca de 0,0002 milissegundos para encontrar uma solução, enquanto o método do polítopo levou cerca de 1,34 segundos. As simulações usando força bruta, gerando 20 milhões de pontos uniformemente distribuídos na

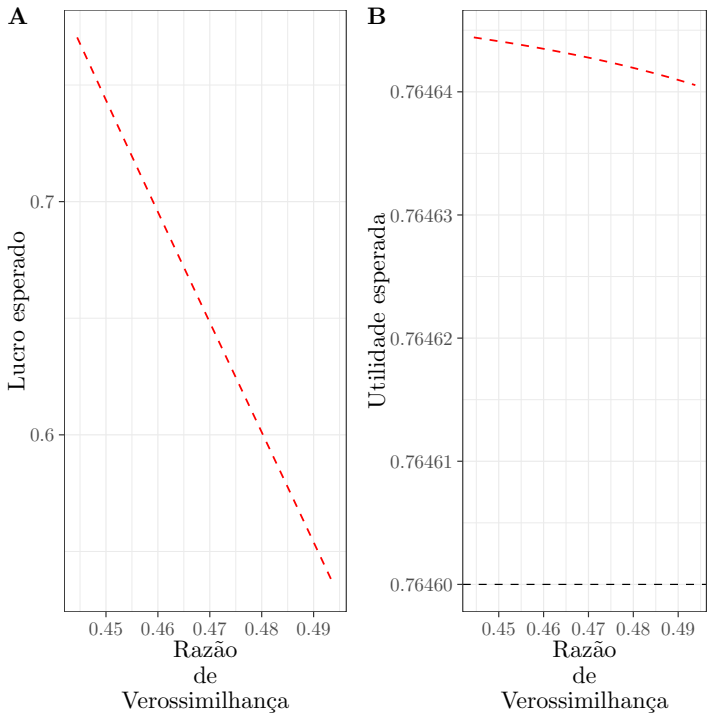
região das restrições, encontrou uma solução próxima dos outros dois algoritmos em 20,34 segundos: 100.000 vezes mais lento do que o método de Newton.

De acordo com Macho-Stadler e Pérez-Castrillo (1997) há uma relação entre as probabilidades e as taxas de salário. A taxa de salário referente ao sucesso w_1 será maior que w_2 quando a razão entre as probabilidades de fracasso e sucesso a depender do esforço, as razões de verossimilhança, forem crescentes:

$$\frac{P(y=0|L=0)}{P(y=0|L=1)} > \frac{P(y=2|L=0)}{P(y=2|L=1)}$$

No caso estudado, a assertiva é verdadeira. O método permite um maior escrutínio do problema: através dele, pode-se fazer uma análise de como o lucro esperado do principal se comporta quanto menor for a razão de probabilidades do maior resultado a depender do esforço. Para analisar isso, o método foi novamente aplicado à função de penalidade, mas ao invés de alterar a taxa de P do problema, mantendo o mesmo constante em 10.000, as probabilidades do problema foram alteradas. Nesse caso a probabilidade de $P(y = 2|L = 1) \in [0,8, 0,9]$, enquanto as probabilidades no caso de um esforço menor foram mantidas constantes.

Figura 9 – Razão de Verossimilhança e Lucro e Utilidade esperadas



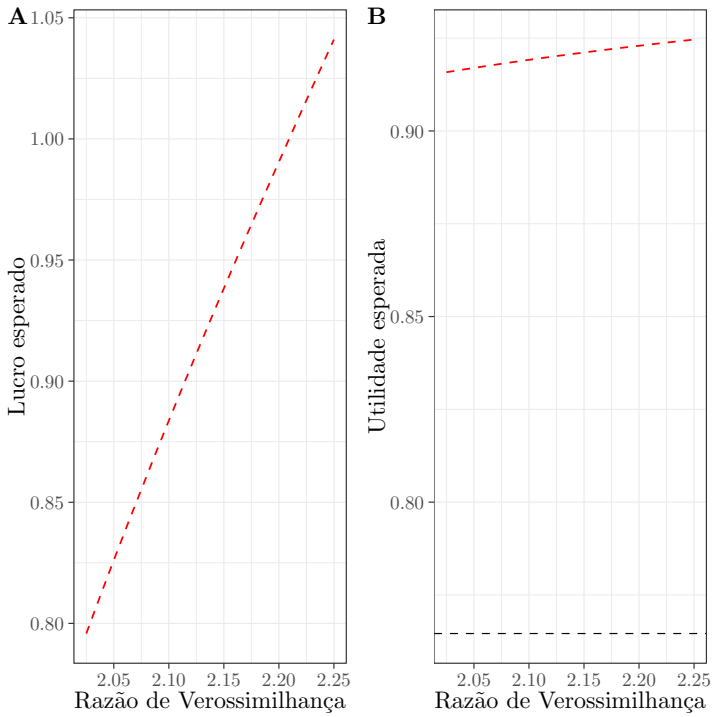
Fonte: Elaborada pelo próprio autor.

Quanto mais provável o resultado $y = 2$ for quando o esforço for $L = 1$, menor será a taxa de verossimilhança. Por consequência, maior será o lucro esperado do principal. Essa relação é observada no gráfico A da figura 9, através da linha tracejada vermelha. No gráfico B,

observamos o que ocorre com a utilidade esperada do agente quando a razão de verossimilhança aumenta. Sua utilidade esperada aumenta quando a razão de verossimilhança é menor. No entanto, como o risco no caso de uma razão de verossimilhança baixa é menor, o salário esperado do agente diminui. A linha tracejada preta é o valor da utilidade proporcionada pela alternativa do mercado. Por meio dessa linha é possível inferir que todos os contratos definidos no intervalo (linha tracejada vermelha) são melhores que a alternativa.

Quando as probabilidades são invertidas, no sentido de que seja mais provável um resultado positivo quando o agente emprega menor esforço na execução de suas atividades, a utilidade esperada do agente aumenta quanto maior for a razão de verossimilhança. Além disso, a utilidade esperada do agente tem um aumento extra em decorrência da queda na desutilidade do esforço. Isso ocorre porque o principal passa a remunerar a ação que está associada ao resultado o qual esteja mais interessado. A figura 10 mostra isso.

Figura 10 – Razão de Verossimilhança e Lucro e Utilidade esperadas com inversão de probabilidades



Fonte: Elaborada pelo próprio autor.

4.2 Soluções para o problema de seleção adversa

Como na seção anterior, o problema de seleção adversa solucionado foi baseado no trabalho de Judd (1998, p. 132). No entanto, apesar do autor descrever o problema, a função utilizada na aplicação do método não é revelada. Levando isso em conta, a função de utilidade do

agente da seção anterior foi reaproveitada e modificada de acordo com os pressupostos teóricos do problema de seleção adversa. O código empregado para a construção do problema bem como sua solução está disponível no apêndice.

O emprego do método de Newton nesse problema gerou matrizes hessianas singulares. Como o problema em questão envolve o uso de muitas variáveis, o método do polítopo foi utilizado para encontrar soluções do problema. Os parâmetros referentes a proporção de agentes de baixo e alto risco (ou ruins e bons) foram mantidas constantes $\theta^R = 0.1$. Nesse caso o problema trata de um planejador social que não quer ter prejuízos, a função a ser maximizada é a média ponderada do bem-estar dos agentes ruim e bom. O valor escolhido foi $\lambda = 0.1$. A tabela a seguir descreve as soluções encontradas ao variar a penalidade de 10 a 1.000.000 com agentes que possuem probabilidades iguais de receberem uma doação por parte do planejador social.

Tabela 2 – Resultados da aplicação do método do polítopo no problema de Seleção Adversa

y_1^R	y_2^R	y_1^B	y_2^B	VC^R	VC^B	Lucro	P
0,8194209	0,8194235	0,8194217	0,8194140	0,0000005	-0,0000005	-0,0194203	10
0,7796150	0,9011808	0,8007373	0,8069751	-0,0001210	0,0001210	-0,0021792	100
0,7764846	0,9028225	0,8094214	0,7619904	-0,0000978	0,0000978	-0,0001169	1.000
0,7529506	1,0124966	0,8155356	0,7197716	0,0001303	-0,0001303	0,0027695	10.000
0,7775562	0,8977505	0,7895277	0,8409527	-0,0000017	0,0000017	0,0000090	100.000
0,7783007	0,8936007	0,7883367	0,8458984	0,0000000	-0,0000000	-0,0000002	1.000.000

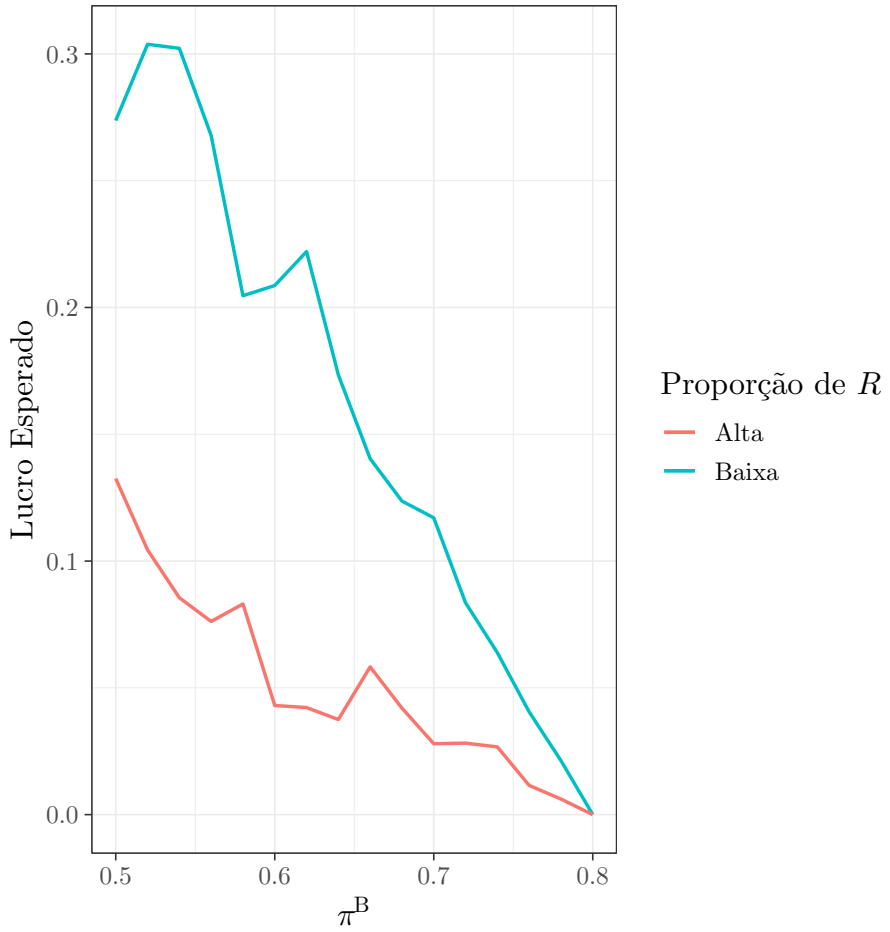
Fonte: Elaborada pelo próprio autor.

Na tabela 2, o sobrescrito $R\ ou\ B$ denota o contrato para cada agente. R é o agente com maior probabilidade de receber doações de maior valor e B é o agente que tem menor probabilidade de receber doações de maior valor. VCI é a violação da restrição de compatibilidade, ou seja, se o agente em questão tem menor utilidade ao consumir contratos destinados a ele. Como nesse caso os agentes são praticamente iguais em termos de risco, os contratos são bastante similares para ambos os agentes e as restrições são minimamente violadas (valores próximos de zero). As pequenas violações sugerem que os contratos relevantes para a solução do problema estão próximos dos resultados encontrados. Na definição original do problema, o autor também encontrou soluções próximas da região (violações mínimas), mas a replicação é difícil devido a dificuldade em precisar quais parâmetros foram utilizados.

O próximo passo foi analisar como os contratos se comportaram ao passo que a probabilidade do agente tipo B receber uma doação (π^B) aumenta. O parâmetro da função de penalidade foi fixado em 100.000. Outro fator analisado foi o comportamento das violações de compatibilidade ao passo que a proporção de agentes do tipo R passou de 0.1 a 0.75. Como o único parâmetro que muda ao longo dos testes é π^B, os gráficos a seguir tomam como referência a influência da variação desse parâmetro.

A medida que a probabilidade do agente tipo B receber uma doação alta (π^B) diminui, maior é o lucro esperado do principal como observado na figura 11. Uma outra característica importante do problema é que quanto maior for a proporção de agentes do tipo R, menor será o lucro esperado do principal.

Figura 11 – Lucro esperado e probabilidade do agente B receber doações altas

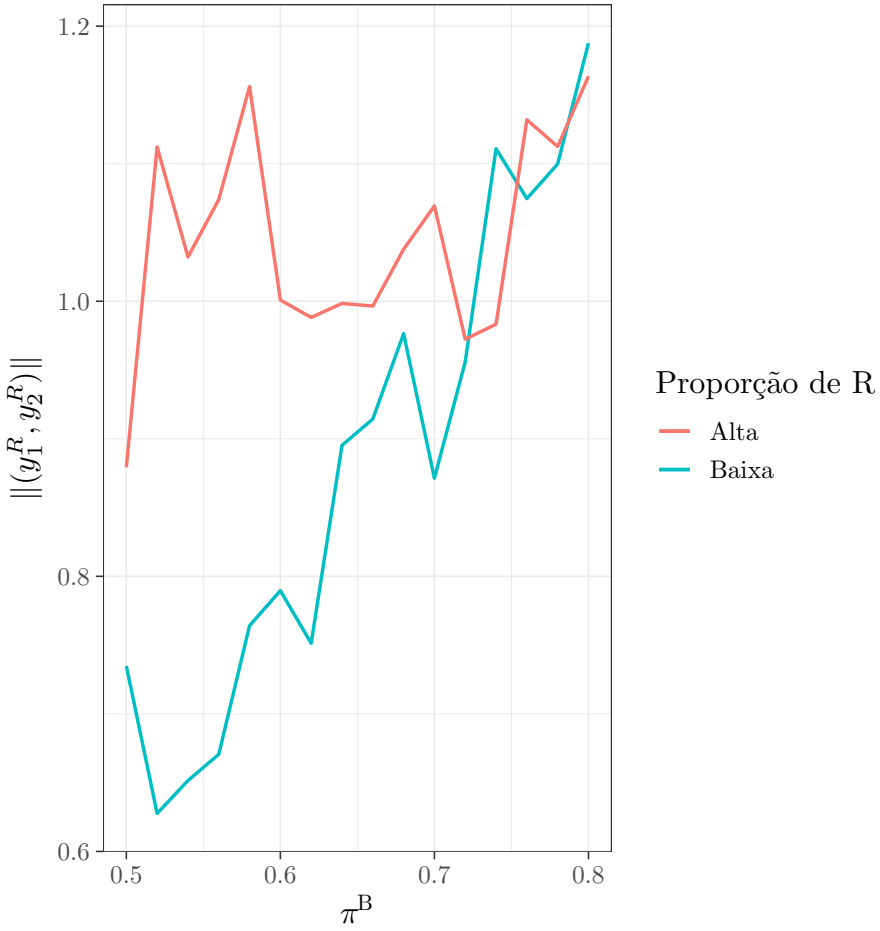


Fonte: Elaborada pelo próprio autor.

Nos resultados encontrados, o agente R só tem incentivos para escolher o contrato destinado a ele (y_1^R, y_2^R) quando a proporção desse tipo de agente é pequena. Caso contrário, essa restrição nunca é respeitada, sugerindo que não exista contrato eficiente para esse agente nessa circunstância. Esse resultado é previsto pela teoria de seleção adversa, como sugerido por Judd (1998), Rothschild e Stiglitz (1976) e Macho-Stadler e Pérez-Castrillo (1997).

A diminuição de π^B possui outro efeito importante sobre o comportamento do agente R. Os valores de y_1^R e y_2^r caem conforme π^B cai. Esse efeito é mais pronunciado quando a proporção de agentes do tipo R é pequena. O mesmo efeito é observado no trabalho de Judd (1998), apesar da possível diferença entre as formas funcionais utilizadas. Para resumir esses contratos, a norma deles foi computada e comparada a π^B na figura a seguir.

Figura 12 – Contratos do agente tipo R e probabilidade do agente tipo B receber doações altas



Fonte: Elaborada pelo próprio autor.

5 Considerações Finais

O desenvolvimento do trabalho em questão descreveu alguns dos problemas de principal-agente e suas respectivas soluções analíticas. Em seguida, possíveis métodos numéricos que solucionassem e permitissem um maior escrutínio desses problemas foram discutidos, expostos e implementados computacionalmente. Por fim, na seção de resultados, das possíveis aplicações desses métodos foram executadas.

A metodologia empregada se mostrou suficiente para solucionar os problemas propostos, como no caso do problema de risco moral, ou encontrar valores que estavam na região da solução, como no caso do problema de seleção adversa. No caso do problema de risco moral, o método de Newton foi o mais rápido para encontrar soluções, mas não foi capaz de encontrar soluções para o problema de seleção adversa devido ao surgimento de matrizes hessianas singulares. O método do polítopo se mostrou como a alternativa de solução do problema. Em ambos casos, resultados bastante similares ao trabalho de Judd (1998) foram encontrados, bem como as implicações teóricas da formalização analítica. Os apêndices contém todos os códigos utilizados da linguagem R, empregada para a solução dos problemas.

Evidentemente, todas as possibilidades acerca do terma não foram esgotadas. Por exemplo, os problemas analisados neste trabalho levaram em conta concavidade estrita tanto para o caso do agente quanto o caso do principal. Jogos de sinalização não foram estudados tampouco suas possíveis soluções, ainda que esse tipo de problema esteja relacionado ao estudo de jogos com informação assimétrica. O trabalho também não levou em conta problemas nos quais agentes possuem, ao mesmo tempo, informação assimétrica e decisões privadas, denominados problemas de principal agente generalizados. Para um estudo desses problemas e suas respectivas soluções numéricas, sugere-se a leitura do trabalho de Dang (2017). Nesse trabalho, o autor utiliza métodos de discretização para avaliar funções contínuas, o que, por exemplo, permitiria avaliar funções de esforço contínuas que não foram tratadas nesse trabalho.

Ainda assim, o que foi desenvolvido aqui permite que outros problemas de classe e estrutura similar possam ser resolvidos a depender dos parâmetros. É importante ressaltar que nas ocasiões em que o método de Newton não funcione, o método do polítopo se mostra como uma alternativa eficiente. O método de força bruta, no entanto, é ideal apenas para a análise de sensibilidade dos parâmetros de uma função, dada a sua velocidade menor e ser mais computacionalmente intensivo.

Referências

AKERLOF, G. A. The market for "lemons": Quality uncertainty and the market mechanism. *Quarterly Journal of Economics*, v. 84, p. 488, 1970. ISSN 0033-5533. Citado na página 22.

CARTER, R. G. Numerical experience with a class of algorithms for nonlinear optimization using inexact function and gradient information. *SIAM J. Scientific Computing*, v. 14, n. 2, p. 368–388, 1993. Citado na página 30.

DANG, N. *Numerical Solutions to Principal-Agent Problems*. Dissertação (Master of Sciences) — Delft University of Technology, 2017. Citado na página 43.

FIANI, R. Teoria dos custos de transação. In: ____. *Economia Industrial*. [S.l.]: Elsevier, 2012. cap. 13, p. 171–181. ISBN 8535263683. Citado 2 vezes nas páginas 17 e 18.

FUDENBERG, D.; TIROLE, J. *Game Theory*. The MIT Press, 1991. ISBN 0262061414. Disponível em: <https://www.ebook.de/de/product/3241100/drew_fudenberg_jean_tirole_game_theory.html>. Citado na página 15.

GIBBONS, R. *A Primer in Game Theory*. Pearson Higher Education, 1992. ISBN 0745011594. Disponível em: <https://www.ebook.de/de/product/3247018/robert_gibbons_a_primer_in_game_theory.html>. Citado 2 vezes nas páginas 15 e 16.

GROSSMAN, S. J.; HART, O. D. An analysis of the principal-agent problem. *Econometrica*, v. 1, p. 7–45, 1983. Citado 2 vezes nas páginas 18 e 19.

JUDD, K. L. *Numerical Methods in Economics*. MIT Press, 1998. ISBN 0262100711. Disponível em: <https://www.ebook.de/de/product/3738332/kenneth_l_judd_numerical_methods_in_economics.html>. Citado 9 vezes nas páginas 13, 25, 26, 30, 32, 37, 39, 41 e 43.

LINGE, S.; LANGTANGEN, H. P. *Programming for Computations - Python*. [S.l.]: Springer International Publishing, 2016. Citado na página 35.

MACHINA, M. J. Choice under uncertainty: Problems solved and unsolved. *Journal of Economic Perspectives*, v. 1, p. 121–154, 1987. ISSN 0895-3309. Citado na página 15.

MACHO-STADLER, I.; PéREZ-CASTRILLO, J. D. *An Introduction to the Economics of Information: Incentives and Contracts*. Oxford University Press, 1997. ISBN 9780198774679. Disponível em: <https://www.amazon.com/Introduction-Economics-Information-Incentives-Contracts-ebook/dp/B004E0Z3S6?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B004E0Z3S6>. Citado 5 vezes nas páginas 17, 19, 22, 38 e 41.

NASH, J. C. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. 2. ed. [S.l.]: Taylor & Francis US, 1990. ISBN 9780852743195. Citado 2 vezes nas páginas 31 e 32.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. *The Computer Journal*, v. 7, n. 4, p. 308–313, 1965. Citado 2 vezes nas páginas 30 e 32.

PRESS, W. H. et al. *Numerical Recipes*. Cambridge University Pr., 2007. ISBN 0521880688. Disponível em: <https://www.ebook.de/de/product/6504686/william_h_press_saul_a_teukolsky_william_t_vetterling_brian_p_flannery_numerical_recipes.html>. Citado na página 32.

ROTHSCHILD, M.; STIGLITZ, J. *Equilibrium in competitive insurance markets*: An essay on the economics of imperfect information. Cambridge, Mass.: MIT Press, 1976. 629-649 p. Citado 2 vezes nas páginas 22 e 41.

VARIAN, H. R. *Intermediate Microeconomics: A Modern Approach (Ninth Edition)*. W. W. Norton & Company, 2014. 226 - 230 p. ISBN 9780393123968. Disponível em: <https://www.amazon.com/Intermediate-Microeconomics-Modern-Approach-Ninth/dp/0393123960?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0393123960>. Citado 2 vezes nas páginas 16 e 17.

WICKHAM, H. *tidyverse: Easily Install and Load the 'Tidyverse'*. [S.l.], 2017. R package version 1.2.1. Disponível em: <https://CRAN.R-project.org/package=tidyverse>. Citado na página 33.

Apêndices

APÊNDICE A – Código para a solução dos problemas de risco moral

Códigos do programa R para a construção dos algoritmos utilizados na solução de problemas.

```r
# Definindo o problema de risco moral

DisutilityJudd <- function(L) {
  if(L == 0){
    return(0)
  } else{
    return(0.1)
  }
}

#Utilidade do agente
WageUtilityFunc <- function(w){
  -exp(-2*w) + 1
}

# Utilidade do agente levando em conta sua desutilidade

AgentUtility <- function(w, L){
  WageUtilityFunc(w) - DisutilityJudd(L)
}

ExpectedUtilBest <- function(w1, w2, prob = c(0.8, 0.2, 0.4, 0.6)){
  y <- prob[1] * AgentUtility(w1, 1) + prob[2] * AgentUtility(w2, 1)
  return(y)
}

ExpectedUtilWorst <- function(w1, w2, prob = c(0.8, 0.2, 0.4, 0.6)){
  y <- prob[3] * AgentUtility(w1, 0) + prob[4] * AgentUtility(w2, 0)
  return(y)
}

ExpectedUtilityDifference <- function(w1, w2, prob = c(0.8, 0.2, 0.4, 0.6)){
  ExpectedUtilBest(w1, w2, prob) - ExpectedUtilWorst(w1, w2, prob)
}
```

Definição do algoritmo do método de Newton.

```r
  metodo_newton_prob <-  function(chute,
  probabilidades = c(0.8, 0.2, 0.4, 0.6),
  p = 100000,
  n = 1000,
  epsilon = 1e-04,
  delta = epsilon){

  #Penalidade
    PenalFunct <- function(w1, w2, prob = probabilidades, P = p, a = 0.7646647167){

```

```
      if(length(w1) > 1){
        w2 <- w1[2]
        w1 <- w1[1]
      }

      y <- (prob[1] * (2 - w1) - prob[2] * w2 -
      P * (pmax(0, -1 * (ExpectedUtilBest(w1, w2, prob) - a)))^2 -
      P * (pmax(0, -1 * ExpectedUtilityDifference(w1, w2, prob)))^2)
      return(y)
    }

    xk <- matrix(chute, byrow = TRUE)
    k <- list(xk)
    for(i in 1:n){
      hessiana <- rootSolve::hessian(f = PenalFunct, x = chute, centered = TRUE)
      gradiente.transposta <- -1 * t(rootSolve::gradient(f = PenalFunct,
      x = chute,
      centered = TRUE))

      # resolvendo o sistema linear

      sk <- qr.solve(hessiana, gradiente.transposta)
      xk.1 <- xk + sk
      k[[i]] <- c(xk.1)

      # Calculando o primeiro teste

      first.test <- (norm(xk - xk.1) < epsilon * (1 + norm(xk)))
      g <- first.test
      if(first.test == TRUE){
        # calculando o segundo teste para verificar a soluc.
        second.test <- (norm(gradiente.transposta) <= delta * (1 + abs(PenalFunct(xk))))
        h <- second.test
        if(second.test == TRUE){
          root.approx <- unlist(tail(k, n = 1))
          names(root.approx) <- c('x', 'y')
          return(root.approx)
        }
      }
      # Chutes da prox. iter.
      xk <- xk.1

      chute <- c(xk.1)
    }
  }

  resultados <- metodo_newton_prob(c(0,0))
```

Resolvendo o mesmo problema com o método do polítopo.

```
# simplex para n vars.

library(tidyverse)

# Calcula o tamanho do politopo; utilizada no processo de contrac.


calculate_size <- function(data){
size <- 0
```

```
for(i in 1:nrow(data)){
vector_size <- data[i, ] - data[nrow(data),]
size <- size + norm(vector_size, type = '2')
}
return(size)
}

# Verifica externamente se o resultado chegou na cond. de termino; poupa iter.

evaluate_function <- function(list, func){
  std <- reduce(list, bind_rows) %>%
  mutate(values = pmap_dbl(., func)) %>%
  .$values %>%
  sd()
  return(std)
}



# Politopo

simplex <- function(vertices, func){

# Calcula os valores de f(x) e os reordena

  init_mat <- reduce(vertices, bind_rows) %>%
    mutate(values = pmap_dbl(., func)) %>%
    arrange(desc(values)) %>%
    mutate(id = row_number())

  if(sd(init_mat$values, na.rm = TRUE) <= 0){
    return(init_mat[, 1:(ncol(init_mat)-2)])
  }else{
    centroid <- colSums(init_mat[1:(nrow(init_mat)-1), 1:(ncol(init_mat)-2)]) *
    (1/(nrow(init_mat)-1))
    reflection <- c(centroid + (centroid - c(filter(init_mat, id == max(id)),
    recursive = TRUE)[1:(ncol(init_mat)-2)]), id = nrow(init_mat) + 1)

# Calcula os valores da reflec. e os reordena
    evaluate <- bind_rows(init_mat, reflection) %>%
      mutate(values = pmap_dbl(.[, 1:(ncol(init_mat)-2)], func)) %>%
      arrange(desc(values)) %>%
      select(-values)

    if(between(which(evaluate$id == max(evaluate$id)), 2, (max(evaluate$id) - 2))){

      return(transpose(select(evaluate, -id)[1:(nrow(init_mat)), ]))

    }else if(which(evaluate$id == max(evaluate$id)) == 1){

      expansion <- c(centroid + 2 * (c(filter(evaluate, id == max(id)),
      recursive = TRUE)[1:(ncol(evaluate)-1)] -
      centroid), id = nrow(evaluate)+1)

      evaluate_test <- bind_rows(evaluate, expansion) %>%
      mutate(values = pmap_dbl(.[, 1:(ncol(evaluate)-1)], func)) %>%
      arrange(desc(values))

      if(which(evaluate_test$id == max(evaluate_test$id)) < which(evaluate_test$id == (max(
          evaluate_test$id)-1))){
```

```
69          return(transpose(select(evaluate_test, -(id:values))[1:(nrow(evaluate_test)-2), ]))
70        }else{
71          return(transpose(select(evaluate, -(id))[1:(nrow(evaluate)-1), ]))
72        }
73
74      }else if(which(evaluate$id == max(evaluate$id)) == (max(evaluate$id) - 1)){
75        contraction <- c(centroid + 0.5*(c(filter(evaluate, id == (max(id)-1)),
76        recursive =TRUE)[1:(ncol(evaluate)-1)] - centroid),
77        id = nrow(evaluate) + 1)
78
79        evaluate <- bind_rows(evaluate, contraction) %>%
80        mutate(values = pmap_dbl(.[, 1:(ncol(evaluate)-1)], func)) %>%
81        arrange(desc(values)) %>%
82        select(-values)
83
84        contraction_test <- near(calculate_size(select(evaluate, -id)[1:nrow(init_mat),]) -
85        calculate_size(select(init_mat, -(id:values))[1:nrow(init_mat),]), 0)
86
87        if((which(evaluate$id == (nrow(evaluate))) > which(evaluate$id == (nrow(evaluate)-2))) &
88        contraction_test != TRUE){
89
90          best_vec <- c(select(evaluate, -id)[1, ], recursive = TRUE)
91          best <- evaluate[1, 1:(ncol(evaluate)-1)]
92          for(i in 1:(nrow(evaluate)-3)){
93            shrink <- best_vec + 0.5*(c(evaluate[i+1, 1:(ncol(evaluate)-1)], recursive = TRUE) -
                  best_vec)
94            best <- bind_rows(best, shrink)
95          }
96          return(transpose(best))
97
98        }else{
99          return(transpose(select(evaluate, -id)[1:(nrow(init_mat)), ]))
100       }
101     }else{
102       contraction <- c(centroid + 0.5*(c(filter(evaluate, id == (max(id)-1)),
103       recursive =TRUE)[1:(ncol(evaluate)-1)] - centroid),
104       id = nrow(evaluate)+1)
105
106       evaluate <- bind_rows(evaluate, contraction) %>%
107       mutate(values = pmap_dbl(.[, 1:(ncol(evaluate)-1)], func)) %>%
108       arrange(desc(values)) %>%
109       select(-values)
110
111       contraction_test <- near(calculate_size(select(evaluate, -id)[1:nrow(init_mat),]) -
112       calculate_size(select(init_mat, -(id:values))[1:nrow(init_mat),]), 0)
113
114       if(which(evaluate$id == (max(evaluate$id)-2)) > which(evaluate$id == max(evaluate$id)) &
115       contraction_test != TRUE){
116         return(transpose(select(evaluate, -id)[1:(nrow(init_mat)), ]))
117       }else{
118         best_vec <- c(select(evaluate, -id)[1, ], recursive = TRUE)
119         best <- evaluate[1, 1:(ncol(evaluate)-1)]
120         for(i in 1:(nrow(evaluate)-3)){
121         shrink <- best_vec + 0.5*(c(evaluate[i+1, 1:(ncol(evaluate)-1)], recursive = TRUE) -
              best_vec)
122         best <- bind_rows(best, shrink)
123       }
124       return(transpose(best))
125       }
126     }
```

```
  }
}


# Calculando politopo
a <- c(0, 0)
b <- c(0, 1)
c <- c(1, 0)

names(a) <- c("w1", "w2")
names(b) <- c("w1", "w2")
names(c) <- c("w1", "w2")

list_of_vectors <- list(a, b, c)

# Observe que as func. de penalidade e de defin. do problema precisam estar no ambiente


n = 1000
solutions <- list()

for(i in 1:n){
  list_of_vectors <- simplex(list_of_vectors, PenalFunct)
  solutions[[i]] <- list_of_vectors
  if(evaluate_function(solutions[[i]], PenalFunct) == 0){
  break
  }
}
```

Resolvendo através do método de força bruta.

```
  library(tidyverse)


  set.seed(330) # reproducibilidade garantida

  lista <- list()
  n = 1
  # i de 1 a 7 garante p entre [10, 10^7]

  for(i in 1:7){
    w1 <- runif(n, min = 0, max = 2)
    w2 <- runif(n, min = 0, max = 2)

    # criando tibble com restric.

    data <- tibble(w1, w2) %>%
      mutate(
        lucro = (0.8*(2-w1)+0.2*(-w2)),
        VI = map2_dbl(w1, w2, .f = ExpectedUtilityDifference),
        VR =map2_dbl(w1, w2, .f = function(w1, w2) ExpectedUtilBest(w1, w2) - AgentUtility(1,
            1))
      ) %>%
      filter((near(VR, 0, tol = 1e-7) | VR > 0), (near(VR, 0, tol = 1e-7) | VI > 0)) %>%
      arrange(desc(lucro))

    lista[[i]] <- head(data, n = 1)
    n = n * 10
  }
```

```
28
29  dados <- reduce(lista, bind_rows)
```

Avaliando o efeito de mudança nas probabilidades sobre os resultados ótimos. Funções de definição do problema e método de Newton devem estar definidas no ambiente. O interessado em reproduzir resultados deve executar o código cuidadosamente devido ao possível surgimento de singularidades.

```
1   # gerando vetores de probabilidades
2
3   vetor_prob1 <- seq(from = 0.81, to = 0.9, by = 0.005)
4   vetor_prob2 <- abs(vetor_prob1 - 1)
5
6   # probabilidades de menor L mantidas
7
8   tabela_probabilidades <- tibble(a = vetor_prob1, b = vetor_prob2, c = 0.4, d = 0.6)
9
10  tabela_probabilidades <- as.data.frame(tabela_probabilidades)
11
12  lista_probabilidades <- lapply(split(tabela_probabilidades,
13  seq(nrow(tabela_probabilidades))),
14  unlist)
15
16  # transformando lista em tabela
17
18  lista_resultados <- map(lista_probabilidades,
19  ~metodo_newton_prob(chute = c(0,0), .x))
20
21  dadobruto <- lapply(lista_resultados, '[', c(1,2)) %>%
22  setNames(nm = letters[1:12])
23
24  dadocoluna <- bind_rows(!!!dadobruto)
25  rm(dadobruto, lista_probabilidades, lista_resultados)
26
27  # preparando a nova tabela com os dados gerados
28
29  tabela_completa <- tabela_probabilidades %>%
30  add_column(w1 = dadocoluna$x, w2 = dadocoluna$y) %>%
31  mutate(w_esperado_max = (a * AgentUtility(w1, 1) + b * AgentUtility(w2, 1)),
32  w_esperado_min = c * w1 + d * w2,
33  razao_verossi_1 = d / b,
34  razao_verossi_2 = c / a,
35  lucro_esperado = (a * (2 - w1) - b * w2)
36  )
37
38  tabela_probabilidades2 <- tibble(a = 0.4, b = 0.6,
39  c = vetor_prob1,
40  d = vetor_prob2)
41
42  lista_probabilidades2 <- lapply(split(tabela_probabilidades2,
43  seq(nrow(tabela_probabilidades2))),
44  unlist)
45
46  lista_resultados2 <- map(lista_probabilidades2,
47  ~metodo_newton_prob(chute = c(0,0), .x))
48
49  dadobruto <- lapply(lista_resultados2, '[', c(1,2)) %>%
50  setNames(nm = letters[1:12])
```

```
51
52  dadocoluna <- bind_rows(!!!dadobruto)
53
54  tabela_completa <- tabela_probabilidades2 %>%
55  add_column(w1 = dadocoluna$x, w2 = dadocoluna$y) %>%
56  mutate(w_esperado_max = (d * AgentUtility(w1, 0) + c * AgentUtility(w2, 0)),
57  w_esperado_min = c * w1 + d * w2,
58  razao_verossi_1 = d / b,
59  razao_verossi_2 = c / a,
60  lucro_esperado = (c * (2 - w1) - d * w2)
61  )
```

APÊNDICE B – Resolvendo o problema de seleção adversa

O problema de seleção adversa depende somente do método do polítopo. O próximo código auxilia o leitor a definí-lo e depois a replicar resultados.

```
# utilidade do agente bom
high_agent <- function(y, probe = 0.8){
  expected_utility <- probe*(-exp(-2*(y[1]))+1) + (1-probe)*(-exp(-2*(y[2]))+1)
  expected_utility
}

# Utilidade do agente ruim

low_agent <- function(y, probe = 0.8){
  expected_utility <- probe*(-exp(-2*(y[1]))+1) + (1-probe)*(-exp(-2*(y[2]))+1)
  expected_utility
}

# Lucro esperado do principal (planejador social)

principal_profit <- function(y1, y2, y3, y4, proportion = 0.1, pb = 0.8){
  expected_utility <- proportion * (0.8*(1-y1)+0.2*(-y2)) +     (1-proportion)*(pb*(1-y3)
      +(1-pb)*(0-y4))
  expected_utility
}

# bem-estar social

welfare<- function(y1, y2, y3, y4, weight = 0.1, probe = 0.8){
  value <- weight * high_agent(c(y1, y2)) + (1-weight) * low_agent(c(y3, y4), probe)
  value
}


p = 10
values <- list()
for(j in 1:5){

  penalty_function <- function(y1, y2, y3, y4, k = p){

    value <- welfare(y1, y2, y3, y4) +
    -k * pmax(0, -(high_agent(c(y1, y2)) - high_agent(c(y3, y4))))^2 +
    -k * pmax(0, -(low_agent(c(y3, y4)) - low_agent(c(y1, y2))))^2 +
    -k * pmax(0, -(principal_profit(y1, y2, y3, y4)))^2

    value
  }

  # chutes iniciais
  a <- c(1, 0, 0, 0)
  b <- c(0, 1, 0, 0)
  c <- c(0, 0, 1, 0)
```

```
    d <- c(0, 0, 0, 1)
    e <- c(1, 1, 1, 1)

    names(a) <- c('y1', 'y2', 'y3', 'y4')
    names(b) <- c('y1', 'y2', 'y3', 'y4')
    names(c) <- c('y1', 'y2', 'y3', 'y4')
    names(d) <- c('y1', 'y2', 'y3', 'y4')
    names(e) <- c('y1', 'y2', 'y3', 'y4')

    list_of_vectors <- list(a, b, c, d, e)

    n = 10000
    solutions <- list()

    for(i in 1:n){
      list_of_vectors <- simplex(list_of_vectors, penalty_function)
      solutions[[i]] <- list_of_vectors
      if(evaluate_function(solutions[[i]], penalty_function) == 0){
        break
      }
    }
    values[[j]] <- reduce(tail(solutions, n = 1)[[1]], bind_rows)
    p = p * 10
  }

  soluts <- list()

  for(i in 1:length(values)){
    soluts[[i]] <- (values[[i]][1, ])
  }

  soluts <- reduce(soluts, bind_rows)

  # Depois do calculo

  incentive_violation_high <- function(y1, y2, y3, y4){
    return(high_agent(c(y1, y2)) - high_agent(c(y3, y4)))
  }

  incentive_violation_low <- function(y1, y2, y3, y4, probe = 0.8){
    value <- low_agent(c(y3, y4), probe) - low_agent(c(y1, y2), probe)
    return(value)
  }


  # resultados
  dados <- soluts %>%
    mutate(
      VCIR = pmap_dbl(., incentive_violation_high),
      VCIB = pmap_dbl(.,incentive_violation_low),
      Lucro = pmap_dbl(., principal_profit),
      P = 10^row_number()
  )
```

Analisando efeitos de mudança nos parâmetros. Perceba que, como alguns parâmetros foram modificados, em alguns pontos as funções que compõem o problema foram redefinidas.

```
  principal_profit <- function(y1, y2, y3, y4, proportion = 0.1, pb = 0.8){
    expected_utility <- proportion * (0.8*(1-y1)+0.2*(-y2)) +
```

```
    (1-proportion)*(pb*(1-y3)+(1-pb)*(0-y4))
    expected_utility
  }

  risco_menor <- seq(0.8, 0.5, by = -0.02)
  values <- list()
  count <- 0
  for(j in risco_menor){
    count <- count + 1

    penalty_function <- function(y1, y2, y3, y4, k = 100000, probe = j){

      value <- welfare(y1, y2, y3, y4, probe = probe) +
      -k * pmax(0, -(high_agent(c(y1, y2)) - high_agent(c(y3, y4))))^2 +
      -k * pmax(0, -(low_agent(c(y3, y4), probe) - low_agent(c(y1, y2), probe)))^2 +
      -k * pmax(0, -(principal_profit(y1, y2, y3, y4, pb = probe)))^2

      value
    }

    # chutes iniciais
    a <- c(1, 0, 0, 0)
    b <- c(0, 1, 0, 0)
    c <- c(0, 0, 1, 0)
    d <- c(0, 0, 0, 1)
    e <- c(1, 1, 1, 1)

    names(a) <- c('y1', 'y2', 'y3', 'y4')
    names(b) <- c('y1', 'y2', 'y3', 'y4')
    names(c) <- c('y1', 'y2', 'y3', 'y4')
    names(d) <- c('y1', 'y2', 'y3', 'y4')
    names(e) <- c('y1', 'y2', 'y3', 'y4')

    list_of_vectors <- list(a, b, c, d, e)

    n = 10000
    solutions <- list()

    for(i in 1:n){
      list_of_vectors <- simplex(list_of_vectors, penalty_function)
      solutions[[i]] <- list_of_vectors

      if(evaluate_function(solutions[[i]], penalty_function) == 0){
        break
      }
    }

    values[[count]] <- reduce(tail(solutions, n = 1)[[1]], bind_rows)
  }

  soluts <- list()
  for(i in 1:length(values)){
    soluts[[i]] <- (values[[i]][1, ])
  }

  dados <- reduce(soluts, bind_rows) %>%
    mutate(
      probe = risco_menor,
      VCIR = pmap_dbl(., incentive_violation_high),
      VCIB = pmap_dbl(.,incentive_violation_low),
```

```
    Lucro = pmap_dbl(., principal_profit),
    theta_b = 0.1
  )


# proporcao de agentes R alta

principal_profit <- function(y1, y2, y3, y4, proportion = 0.75, pb = 0.8){
  expected_utility <- proportion * (0.8*(1-y1)+0.2*(-y2)) +
  (1-proportion)*(pb*(1-y3)+(1-pb)*(0-y4))
  expected_utility
}

risco_menor <- seq(0.8, 0.5, by = -0.02)
values <- list()
count <- 0

for(j in risco_menor){
  count <- count + 1

  penalty_function <- function(y1, y2, y3, y4, k = 100000, probe = j){

    value <- welfare(y1, y2, y3, y4, probe = probe) +
    -k * pmax(0, -(high_agent(c(y1, y2)) - high_agent(c(y3, y4))))^2 +
    -k * pmax(0, -(low_agent(c(y3, y4), probe) - low_agent(c(y1, y2), probe)))^2 +
    -k * pmax(0, -(principal_profit(y1, y2, y3, y4, pb = probe)))^2

    value
}

  # chutes iniciais
  a <- c(1, 0, 0, 0)
  b <- c(0, 1, 0, 0)
  c <- c(0, 0, 1, 0)
  d <- c(0, 0, 0, 1)
  e <- c(1, 1, 1, 1)

  names(a) <- c('y1', 'y2', 'y3', 'y4')
  names(b) <- c('y1', 'y2', 'y3', 'y4')
  names(c) <- c('y1', 'y2', 'y3', 'y4')
  names(d) <- c('y1', 'y2', 'y3', 'y4')
  names(e) <- c('y1', 'y2', 'y3', 'y4')

  list_of_vectors <- list(a, b, c, d, e)

  n = 10000
  solutions <- list()
  for(i in 1:n){
    list_of_vectors <- simplex(list_of_vectors, penalty_function)
    solutions[[i]] <- list_of_vectors
    if(evaluate_function(solutions[[i]], penalty_function) == 0){
    break
    }
  }

  values[[count]] <- reduce(tail(solutions, n = 1)[[1]], bind_rows)
}

soluts2 <- list()

```

```
123  for(i in 1:length(values)){
124    soluts2[[i]] <- (values[[i]][1, ])
125  }
126
127  dados2 <- reduce(soluts2, bind_rows) %>%
128    mutate(
129      probe = risco_menor,
130      VCIR = pmap_dbl(., incentive_violation_high),
131      VCIB = pmap_dbl(.,incentive_violation_low),
132      Lucro = pmap_dbl(., principal_profit),
133      theta_b = 0.75
134    )
135
136  dados_completos <- bind_rows(dados, dados2) %>%
137    mutate(
138      contrato_r = map2_dbl(y1, y2, ~norm(c(.x, .y), type = '2')),
139      var_cont = (contrato_r / lag(contrato_r) - 1)
140    )
```