UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS
E DE COMPUTAÇÃO CIENTÍFICA

Relatório de Trabalho
SME0211-Otimização Linear

Implementação do Algoritmo Simplex: Abordagem de Duas Fases para Resolução de Problemas de Otimização Linear na Forma Padrão

Professora: Marina Andretta
Aluno: Felipe Destaole **Nº USP:** 13686768
Aluno: João Pedro Farjoun Silva **Nº USP:** 13731319
Aluno: Luís Roberto Piva **Nº USP:** 13687727
Aluno: Téo Sobrino Alves **Nº USP:** 12557192

São Carlos- SP
Dezembro 2023

Conteúdo

1 Introdução

O algoritmo Simplex é uma técnica clássica para resolver problemas de otimização linear, que envolvem maximizar ou minimizar uma função linear sujeita a um conjunto de restrições lineares. Desenvolvido por George Dantzig na década de 1940, o Simplex é amplamente utilizado devido à sua eficácia e aplicabilidade a uma variedade de problemas do mundo real.

Na resolução de problemas de otimização linear na forma padrão, o algoritmo Simplex opera através de uma tabela (tableau) que representa o sistema de equações lineares. Essa tabela inclui informações cruciais sobre as restrições do problema, a função custo associada e as soluções básicas do sistema. O objetivo é iterativamente movimentar-se de uma solução viável básica para outra, buscando otimizar a função objetivo.

2 Objetivo

O objetivo principal do trabalho é desenvolver uma implementação do algoritmo simplex para resolver problemas de otimização linear na forma padrão. A implementação deve abordar a resolução do problema utilizando a técnica de duas fases, utilizando-se da Regra de Pivoteamento de Bland, ambas aprendida em sala de aula.

O trabalho incluirá a resolução de várias instâncias de problemas de otimização linear, escolhidos pelo grupo para teste. Para avaliar a eficácia da implementação, serão reportados os resultados computacionais, incluindo o tempo de solução, o número de iterações realizadas pelo algoritmo simplex e o valor alcançado para a função objetivo. Esses resultados serão obtidos através da execução da implementação nos problemas escolhidos.

Este trabalho teve como propósito empregar nossos conhecimentos adquiridos em sala de aula sobre o funcionamento do método Simplex. Dessa forma, buscamos programar uma das mais robustas ferramentas para a resolução de problemas de otimização linear, aplicando-a e testando-a em diversas instâncias.

3 Metodologia

Dentre as opções de trabalho disponíveis, a decisão de dedicar-nos à codificação foi tomada por consenso unânime, impulsionada pela curiosidade de explorar a programação do Simplex como uma experiência mais empolgante e enriquecedora para a grade da disciplina. Essa escolha fundamenta-se na oportunidade excepcional de aprimorar habilidades de programação e, simultaneamente, aplicar de maneira prática os princípios lógicos subjacentes ao Simplex.

Com a definição do segmento do trabalho, avançamos para a escolha da linguagem de programação mais adequada para implementar o algoritmo simplex. Inicialmente, experimentamos códigos em Python; contudo, optamos pela linguagem C++, respaldados pela robustez e eficiência que ela proporciona, alinhando-se de maneira mais coerente com os requisitos do projeto. Essa decisão reflete não apenas a busca por uma implementação técnica, mas também o desejo de explorar de forma mais aprofundada o potencial da programação na resolução de problemas de otimização linear.

Com o propósito de atingir nossos objetivos, dividimos a metodologia em quatro partes principais, buscando uma organização eficiente para o relatório. A primeira etapa engloba a conversão dos arquivos de teste, originalmente codificados em MPS, para o formato TXT. A segunda fase concentra-se na leitura dos arquivos que o usuário pretende utilizar. A terceira etapa envolve a aplicação do método simplex de duas fases para resolver o problema específico apresentado. E, por fim, chegamos à quarta fase, que consiste na análise dos resultados.

3.1 MPS para TXT

Após a escolha da linguagem, partimos para a fase de codificação. Inicialmente, dedicamos tempo à reflexão sobre a estrutura do arquivo que conteria as informações do problema linear, assim como o formato mais adequado para armazenar esses dados. Optamos por utilizar arquivos .txt como entrada para o algoritmo simplex, devido à sua facilidade de manipulação.

Ao buscar instâncias na biblioteca Netlib para testar nosso código, deparamo-nos com o desafio de que todas as instâncias estavam no formato .mps. Diante dessa situação, desenvolvemos um código capaz de extrair as informações desses arquivos .mps e convertê-las para o formato .txt, viabilizando assim a utilização desses dados como entrada para o simplex.

Inicialmente, para descomprimir os arquivos .mps, empregamos um código em linguagem C disponibilizado pela própria Netlib, initulado **'emeps.c'**. Este código, elaborado por David M. Gay dos Laboratórios Bell da AT&T, tem como finalidade expandir programas MPS comprimidos no formato Netlib para o formato MPS padrão.

Após a descompressão dos arquivos, convertemos os arquivos .mps resultantes para o formato .txt utilizando um script em Python.

Este script em Python carrega um problema de programação matemática no formato MPS usando a biblioteca 'pysmps'. Ele extrai informações essenciais do problema, como tipos de restrições, função de custo, restrições e variáveis. Em seguida, realiza transformações no problema, como a adição de variáveis de folga e a manipulação de restrições, visando a transformação do problema para a sua forma padrão.

As informações modificadas, incluindo o número de variáveis, o número de restrições, a função de custo e a matriz de custo ajustada, são impressas na saída padrão. O script também verifica e ajusta as restrições nas variáveis, se aplicável.

Os arquivos .txt gerados assumirão a seguinte estrutura:

$$\#\text{Nº de restrições} \quad \#\text{Nº de variáveis}$$

$$\begin{array}{cccccc} & c_1 & c_2 & \dots & c_n & \\ A_{11} & A_{12} & \dots & A1n & b_1 \\ A_{21} & A_{22} & \dots & A2n & b_2 \\ & & & \vdots & & \\ A_{m1} & A_{m2} & \dots & A_{mn} & b_m \end{array}$$

onde:

- **m** é o número de restrições;
- **n** de variáveis.

3.2 Leitura dos arquivos

Iniciamos nosso código no arquivo **'main.cpp'**, criando o simplex como uma classe em C++. Uma classe é uma estrutura que encapsula dados e funções relacionadas, proporcionando uma abstração eficiente. A classe simplex contém duas partes distintas: a privada e a pública.

Na parte privada, estão contidos valores e funções que não são acessíveis diretamente pelo usuário, sendo protegidos por sua importância e natureza. Nessa seção do código, incluímos a matriz de restrições do problema, o vetor de custos das restrições, o vetor da função objetivo, a matriz base associada ao problema, os custos reduzidos e o tableau associado. Adicionalmente, implementamos funções como a construção, a atualização e a resolução do tableau.

A parte pública é aquela que o usuário pode manipular. Nessa seção, encontram-se funções como a inicialização do simplex, o solver do simplex, o solver para a primeira e a segunda fase do simplex. Introduzimos também uma classe adicional chamada "Reader", cuja principal função é ler diferentes tipos de arquivos de instâncias, como MPS comprimidos na forma da biblioteca Netlib, MPS descomprimidos e arquivos TXT.

A próxima etapa do código envolve a leitura do arquivo a ser utilizado. Utilizamos a classe "Reader"e suas funções para determinar se o usuário deseja empregar um arquivo MPS comprimido, um MPS descomprimido ou um arquivo TXT.

No caso de um arquivo MPS comprimido, o 'path' fornecido pelo usuário é utilizado para descomprimir o arquivo que ele fornce, conforme o código C previamente comentado em 3.1. Em seguida, transformamos o MPS descomprimido em um arquivo TXT através de nosso script em Python, também explicado anteriormente em 3.1. Com esse TXT e utilizando outra função da classe "Reader"organizamos os dados do problema, como restrições e variáveis, para um TXT final como queríamos.

Para um arquivo MPS descomprimido, enviamos o arquivo fornecido diretamente para nosso código em Python. O código Python retorna um arquivo TXT contendo as informações do problema. Em seguida, utilizamos esse arquivo para organizar os dados do problema, como restrições e variáveis, para um TXT final, adequando-o para uso no código.

No último caso, simplesmente recebemos o arquivo TXT enviado pelo usuário e o utilizamos na função que ajusta os dados do problema no código, garantindo a devida adequação do arquivo.

3.3 Simplex de Duas fases por Bland

Após descomprimir e converter os arquivos para o formato TXT, avançamos para a etapa de aplicação do solver do simplex de duas fases, empregando o método de pivoteamento de Bland para otimizar o processo.

O método de pivoteamento de Bland é uma estratégia fundamental no contexto do Simplex, destinada a prevenir ciclos indesejados. Essa abordagem envolve a escolha da variável de entrada e de saída com base em regras lexicográficas, dando prioridade às variáveis com índices mais baixos. Tal procedimento assegura uma seleção única e determinística das variáveis, evitando loops infinitos durante a resolução do problema de programação linear.

Quanto ao método de duas fases do Simplex, sua aplicação se torna necessária quando o problema inclui restrições de desigualdade e o tableau inicial não satisfaz as condições de factibilidade. Na primeira fase, o foco está em encontrar uma solução viável básica, mesmo que não seja ótima. Posteriormente, a segunda fase consiste na aplicação do Simplex padrão para otimizar a solução obtida na etapa anterior.

No código da primeira fase, iniciamos com a construção do tableau do sistema auxiliar, incorporando as variáveis artificiais e introduzindo uma nova função objetivo que engloba essas variáveis, conforme abordado durante as aulas. Posteriormente, procedemos à resolução e atualização do tableau, aplicando as técnicas aprendidas em sala de aula, agora com ênfase na eficiência computacional.

Caso o problema se revele inviável ao final da primeira fase, o programa é encerrado. No entanto, se o problema for viável, avançamos para a segunda fase do simplex. Nessa etapa, utilizamos o ponto inicial encontrado pela primeira fase e avaliamos se ele é ótimo. Se não for, resolvemos o tableau associado ao problema original, seguindo a regra de Bland.

3.4 Análise dos Resultados

Ao concluir o processo, o programa imprime informações relevantes sobre o problema, como o número de iterações na primeira fase e na segunda fase, e o tempo usado no processo todo. Adicionalmente, exibe o ponto ótimo achado bem como o valor da função objetivo neste ponto, proporcionando uma visão abrangente do desempenho do solver simplex de duas fases no contexto específico do problema em questão.

Para uma análise mais precisa do tempo de execução, optamos por executar o programa em 10 iterações distintas, calculando a média aritmética resultante para cada instância escolhida. Essa abordagem visa fornecer uma representação mais generalizada, estável e confiável dos tempos de execução, evitando depender de valores temporais específicos que podem ser influenciados por variações ocasionais.

Para fins de comparação, localizamos as soluções das instâncias diretamente na biblioteca Netlib online. Em seguida, procedemos à comparação dos valores da função objetivo ao término do método simplex. Esta abordagem nos permitiu avaliar as eficácias relativas das soluções obtidas em relação aos dados de referência disponíveis na Netlib.

Com os valores da função objetivo em mãos, pudemos calcular os erros cometidos utilizando a seguinte fórmula:

$$\frac{|(valor\_obtido - valor\_esperado)|}{|valor\_esperado|} \cdot 100\%$$

4 Resultados

Para a etapa de testagem, selecionamos 11 instâncias provenientes da biblioteca Netlib. As instâncias escolhidas foram as seguintes: 'afiro.mps', 'agg2.mps', 'israel.mps', 'sc50a.mps', 'sc50b.mps', 'sc205.mps', 'scagr25.mps', 'scsd1.mps', 'share2B.mps', 'stadat.mps' e 'stocfor1.mps'. Essa seleção abrange diferentes tipos de problemas e contribuirá para uma avaliação abrangente do desempenho do método em questão.

Estas instâncias foram adquiridas diretamente do site do Netlib e estão em formato .mps comprimido, portanto, estaremos aplicando o primeiro cenário abordado na subseção 3.1.

Como mencionado na subseção 3.4, executamos cada instância em 10 repetições para obter um tempo médio, e calculamos o erro médio entre os valores obtidos com os valores esperados da biblioteca Netlib.

Antes de apresentarmos os resultados obtidos, é fundamental discutir os valores de um parâmetro crucial, o epsilon, que exerce significativa influência sobre os resultados. Este valor assume uma importância notável, visto que no código[1], ele regula a tolerância de erro nas operações de ponto flutuante. Ademais, optamos pela utilização de ponto flutuante devido à minimização do consumo de memória.

Assim sendo, em determinados problemas, optamos por um epsilon de maior magnitude, enquanto em outras instâncias, empregamos um epsilon de menor valor. Vamos dividir as instâncias de acordo com o epsilon usado.

4.1 Para: $\varepsilon = 1 \times 10^{-5}$

AFIRO	
nº de Variáveis	nº de Restrições
32	28
Valor função objetivo(esperada)	
-464.75314286	
Valor função objetivo(obtida)	
-464.753	
Tempo Médio	Erro (em %)
0.0018911281	0.00003%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
515	5

[1]No script 'simplex.cpp', na linha 208, podemos trocar os valores desse epsilon.

SC50A	
nº de Variáveis	nº de Restrições
48	51
Valor função objetivo(esperada)	
-64.575077059	
Valor função objetivo(obtida)	
-64.5751	
Tempo Médio (s)	Erro (em %)
0.004654458	0.00096%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
569	1

SC50B	
nº de Variáveis	nº de Restrições
48	51
Valor função objetivo(esperada)	
-70.000000000	
Valor função objetivo(obtida)	
-70	
Tempo Médio (s)	Erro (em %)
0.0046914	0%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
519	1

SCAGR25	
nº de Variáveis	nº de Restrições
500	472
Valor função objetivo(esperada)	
-14753433.061	
Valor função objetivo(obtida)	
-14755200.00	
Tempo Médio (s)	Erro (em %)
44.8314	0.011976%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
30368	1072

ISRAEL	
nº de Variáveis	nº de Restrições
142	175
Valor função objetivo(esperada)	
-896644.82186	
Valor função objetivo(obtida)	
-897693	
Tempo Médio (s)	Erro (em %)
3.4864825	0.1169%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
27960	1860

AGG2	
nº de Variáveis	nº de Restrições
302	517
Valor função objetivo(esperada)	
-20239252.356	
Valor função objetivo(obtida)	
-20239300	
Tempo Médio (s)	Erro (em %)
25.0768	0.0002354%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
12381	319

4.2 Para: $\varepsilon = 1 \times 10^{-2}$

SCSD1	
nº de Variáveis	nº de Restrições
760	78
Valor função objetivo(esperada)	
-8.6666666743	
Valor função objetivo(obtida)	
-8.66671	
Tempo Médio (s)	Erro (em %)
7.8335322	0.00049991%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
46810	5040

SHARE2B	
nº de Variáveis	nº de Restrições
79	97
Valor função objetivo(esperada)	
-415.73224074	
Valor função objetivo(obtida)	
-415.728	
Tempo Médio (s)	Erro (em %)
0.1404643	0.001020%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
4877	113

SC205	
nº de Variáveis	nº de Restrições
203	206
Valor função objetivo(esperada)	
-52.202061212	
Valor função objetivo(obtida)	
-52.2031	
Tempo Médio (s)	Erro (em %)
1.636591	0.00198993%
nº de iterações(primeira fase)	nº de iterações(segunda fase)
7889	1

STADATA	
nº de Variáveis	nº de Restrições
1405	490
Valor função objetivo(esperada)	
1257.6995	
Valor função objetivo(obtida)	
1109.57	
Tempo Médio (s)	Erro (em %)
22.7586	$\sim 12\%$
nº de iterações(primeira fase)	nº de iterações(segunda fase)
17424	1086

STOCFOR1	
nº de Variáveis	nº de Restrições
165	117
Valor função objetivo(esperada)	
-41131.976219	
Valor função objetivo(obtida)	
-45172.7	
Tempo Médio (s)	Erro (em %)
0.143881	0,098238 %
nº de iterações(primeira fase)	nº de iterações(segunda fase)
7700	200

5 Discussão dos Resultados

Os resultados obtidos da execução do algoritmo simplex em diversas instâncias demonstram um desempenho significativo em relação à precisão e eficiência computacional. Neste contexto, enfatizaremos aspectos relevantes e discutiremos observações oriundas da análise desses resultados:

5.1 Precisão na Função Objetivo

Nos testes realizados, destacou-se a notável precisão da função objetivo obtida pelo algoritmo simplex em relação aos valores esperados da biblioteca Netlib, geralmente mantendo-se abaixo de 10%. A análise dos erros médios revelou resultados consistentes e precisos, especialmente em instâncias de menor porte. Observou-se que, para instâncias mais complexas e extensas, o algoritmo manteve um desempenho satisfatório, com erros médios que, embora maiores em comparação com instâncias menores, não se mostraram exponencialmente elevados.

Além disso, notamos que a variação nos erros médios foi influenciada por diversos fatores, incluindo o tamanho da instância e sua complexidade. Instâncias de grande porte e alta complexidade tendem a demandar um maior número de iterações, o que pode contribuir para uma ligeira elevação nos erros médios. Contudo, a implementação eficaz da Regra de Bland na escolha das variáveis de entrada e saída, aliada à seleção adequada do parâmetro ε, foi crucial para mitigar possíveis impactos adversos nos resultados.

5.2 Números de iterações

A variação significativa no número de iterações observado durante a execução do algoritmo simplex reflete a diversidade de complexidade entre as diferentes instâncias testadas. Em particular, instâncias como "SCAGR25"destacam-se ao demandar um número considerável de iterações. Esse cenário sugere que determinados problemas apresentam complexidade diferentes, exigindo um número substancial de passos para atingir a solução ótima.

Além disso, é crucial destacar que o número de iterações está relacionado ao método utilizado, no caso, o simplex de duas fases. Nesse contexto, o algoritmo enfrenta o desafio de resolver dois tableau, cada um correspondendo a um problema distinto. Essa

abordagem explora a dualidade entre as fases, uma para encontrar uma solução viável inicial e outra para otimizar essa solução em direção ao ótimo. A interação entre essas fases adiciona uma camada de complexidade ao processo, influenciando diretamente o número total de iterações necessário para alcançar a solução final.

5.3 Sensibilidade do Parâmetro ε

A análise da sensibilidade do algoritmo ao parâmetro ε, que regula a tolerância de erro nas operações de ponto flutuante, revelou pontos interessantes. A redução dos valores de ε resultou em erros percentuais ainda menores, evidenciando a importância de calibrar esse parâmetro de acordo com as características específicas do problema. Notavelmente, em problemas mais complexos e de maiores dimensões, a escolha de ε menor foi preferida, pois isso minimizou a disparidade entre os valores obtidos e aqueles observados na biblioteca Netlib.

Adicionalmente, observamos que a utilização de ε menor em problemas mais extensos e complexos poderia levar o algoritmo a classificar o problema como "unbounded"(ilimitado). Isso destaca a necessidade de uma cuidadosa seleção de ε, pois valores excessivamente pequenos podem impactar adversamente a interpretação da factibilidade do problema, influenciando a validade das soluções obtidas.

5.4 Tempo de Execução

A relação entre o tempo de execução do algoritmo simplex, o tamanho e a complexidade das instâncias é diretamente proporcional, como era de se esperar. Problemas de maior envergadura e complexidade demandaram um maior número de iterações, resultando em um tempo de execução proporcionalmente mais extenso. A eficiência do algoritmo simplex implementado se manifesta nos tempos médios de execução, que se mostraram relativamente curtos para a maioria das instâncias testadas. Entretanto, mesmo em cenários mais desafiadores, exemplificados pelas instâncias "ISRAEL"e "SCAGR25", os tempos de execução se mantiveram razoáveis.

6 Conclusão

Em conclusão, a análise aprofundada do algoritmo simplex para resolver problemas de programação linear revelou sua eficácia e sensibilidade em uma variedade de situações. A precisão na obtenção da função objetivo foi robusta, mantendo erros médios consistentemente abaixo de 10%, destacando a importância da implementação cuidadosa da Regra de Bland e da escolha adequada do parâmetro ε.

Ao lidar com instâncias mais complexas, o algoritmo manteve um desempenho satisfatório, mesmo que os erros médios tenham sido ligeiramente mais elevados. O número de iterações variável entre diferentes casos refletiu a diversidade de complexidade nos problemas de programação linear, sendo a abordagem de duas fases do simplex eficaz para lidar com restrições complexas, embora tenha aumentado o número total de iterações.

A sensibilidade do algoritmo ao parâmetro ε ressaltou a necessidade de adaptação desse valor às características específicas do problema, buscando um equilíbrio entre erros

percentuais menores e a prevenção de classificações incorretas. O tempo de execução, proporcional à complexidade das instâncias, manteve-se razoável, evidenciando a eficiência do algoritmo simplex implementado em diferentes cenários.

Em resumo, este trabalho proporcionou uma compreensão abrangente do algoritmo simplex, suas nuances de implementação e seu desempenho. As análises e discussões apresentadas contribuem significativamente para uma compreensão mais profunda desse método fundamental na otimização linear, destacando sua viabilidade prática em diversas situações.

7 Referências

Netlib. (n.d.). LP Data README. Netlib. Disponível em: https://www.netlib.org/lp/data/readme. Acesso em: 02 dez. 2023.

Netlib. (n.d.). LP Data. Netlib. Disponível em: https://www.netlib.org/lp/data/index.html. Acesso em: 02 dez. 2023.

Centro Nacional de Processamento de Alto Desempenho (CENAPAD). (Ano desconhecido). Sample MPS Input Files. Disponível em: https://www.cenapad.unicamp.br/parque/manuais/OSL/oslweb/features/feat24DT.htm. Acesso em: 02 dez. 2023.

ANDRETTA, Marina. Implementação do tableau completo. ICMC-USP. Disponível em: https://drive.google.com/file/d/15O7OFvzpU6n3c6nlNmeMRMINh9G6Q2DF/view. Acesso em: 02 dez. 2023.

ANDRETTA, Marina. Regras para evitar ciclagem. ICMC-USP. Disponível em: https://drive.google.com/file/d/1aBnwMal2kB5g0nPBF-m53c63YcbvxGMB/view. Acesso em: 02 dez. 2023.

ANDRETTA, Marina. Solução básica viável inicial. ICMC-USP. Disponível em: https://drive.google.com/file/d/1ZZ1hpYJgjuXaVSFke_iIXFG5PiYjgMXV/view. Acesso em: 02 dez. 2023.