Instituto de Ciências Matemáticas e da Computação
Bacharelado em Sistemas da Informação
SME0510 - Introdução a Pesquisa Operacional

Implementação do Método Simplex

Guilherme Henrique Galdini Tosi - N° USP 11781587
Amália Vitória de Melo - N° USP 13692417

São Carlos, Novembro de 2024

Conteúdo

1 Introdução

O Método Simplex é um algoritmo fundamental para a resolução de problemas de otimização linear, amplamente utilizado na análise de modelos matemáticos que envolvem variáveis e restrições lineares. Sua importância reside na capacidade de encontrar soluções ótimas, ou indicar a impossibilidade de uma solução viável, para problemas que podem ser expressos em uma forma padrão, como maximização ou minimização de uma função objetivo sujeita a um conjunto de restrições lineares.

Este trabalho tem como objetivo a implementação de uma versão do Método Simplex com uma estrutura robusta para a resolução de problemas de programação linear. A implementação inclui otimizações para a seleção de pivôs e mecanismos para o tratamento de situações de degeneração, que podem surgir durante o processo. A degeneração, que ocorre quando múltiplas variáveis podem entrar ou sair da base com o mesmo custo, pode levar a ciclos infinitos, complicando o avanço do algoritmo. Para mitigar esses problemas, a técnica de perturbação numérica (EPSILON) é aplicada, introduzindo pequenas variações nos valores dos coeficientes das funções objetivo e das restrições, com o intuito de quebrar a simetria que gera ciclos infinitos.

Além disso, o trabalho aborda a Fase 1 do Método Simplex, uma abordagem necessária quando o problema apresenta uma solução inicial inviável. A Fase 1 permite a construção de uma solução básica viável antes de proceder à Fase 2, que visa a otimização efetiva do modelo. O método também lida com a possibilidade de problemas de otimalidade ilimitada, onde o valor da função objetivo pode ser aumentado ou diminuído indefinidamente, sem atingir um valor ótimo.

Por fim, são discutidos os potenciais desafios e limitações da implementação, incluindo a necessidade de um gerenciamento eficiente de ciclos infinitos e degeneração, fatores que podem impactar a estabilidade e o desempenho do algoritmo, exigindo soluções adicionais para garantir a precisão e a convergência do método. O uso do Método Simplex, com as melhorias e estratégias descritas, proporciona uma ferramenta eficiente e adaptável para a resolução de problemas de otimização linear complexos.

2 Implementação do Método Simplex para Minimização

O método Simplex é um algoritmo amplamente utilizado para resolver problemas de otimização linear, como problemas de minimização e maximização.

2.1 Formulação do Problema

Um problema de minimização linear pode ser formulado da seguinte maneira:

$$\text{Minimizar} \quad c^T x$$

sujeito às restrições:

$$Ax = b$$

e

$$x \geq 0$$

onde:

- x_j são as variáveis de decisão;
- c_j são os coeficientes da função objetivo;
- A_{ij} são os coeficientes das restrições;
- b_i são os valores do lado direito das restrições.

2.2 Passos da Implementação do Algoritmo Simplex

A implementação do método Simplex envolve os seguintes passos principais:

1. **Transformação para a Forma Padrão:** O problema deve ser transformado na forma padrão, caso necessário. Para isso, pode-se utilizar variáveis de folga para transformar desigualdades em igualdades.

$$\sum_{j=1}^{n} a_{ij} x_j + s_i = b_i \quad \forall i = 1, \ldots, m$$

   onde s_i são as variáveis de folga.

2. **Construção da Tabela Simplex:**

   A tabela Simplex é montada, onde as colunas representam as variáveis de decisão e de folga, e as linhas representam as restrições do problema. A última linha contém os coeficientes da função objetivo.

3. **Cálculo da Função Objetivo:**

   Inicialmente, calcula-se o valor da função objetivo Z a partir da tabela. O valor de Z será atualizado conforme o algoritmo avança, onde, nessa versão do programa, temos que somente os casos ótimos e infinitos terão valores da função objetivo, os casos infactíveis terão valor dado como None.

4. **Seleção da Variável de Entrada:**

   A variável que entrará na base é a que possui o coeficiente mais negativo na linha da função objetivo (para problemas de minimização).

5. **Seleção da Variável de Saída:**

   A variável que sairá da base é escolhida com base no critério da razão mínima:

   $$\frac{b_i}{a_{ij}} \quad \text{onde} \quad a_{ij} > 0$$

   Escolhe-se a variável associada ao menor valor da razão mínima.

6. **Pivoteamento:**

   Após a seleção das variáveis de entrada e saída, realiza-se a operação de pivoteamento, que envolve a atualização da tabela Simplex, de modo que a variável de entrada substitua a variável de saída na base.

7. **Iteração:**

   O algoritmo repete os passos 4 a 6 até que não haja mais coeficientes negativos na linha da função objetivo, indicando que a solução ótima foi alcançada.

8. **Solução Ótima:**

   Quando a função objetivo não pode ser mais reduzida, o algoritmo termina, e a solução ótima é dada pelas variáveis básicas (ou seja, as variáveis que estão na base do sistema).

3 Implementação do Programa Simplex

O código foi implementado em Python, utilizando as bibliotecas NumPy e os para auxiliar na modelagem do problema de otimização linear. Foram também aplicadas estratégias específicas para a seleção de pivôs e controle de degenerescência, detalhadas a seguir.

3.1 Leitura do Arquivo de Problema

Para ler os coeficientes do problema de otimização, foi implementada a função `read_problem_file`, que interpreta, que acessa a pasta 'problems', disponibilizada no arquivo .zip fornecido como material para o projeto(a pasta deve estar no mesmo diretório do programa), a partir disso, a função lê os dados do arquivo, trata-os e faz a organização dos vetores e matrizes de coeficientes:

```
import numpy as np

def read_problem_file(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

    c = []
    A = []
    b = []

    for line in lines:
        stripped_line = line.strip()
        if stripped_line.startswith("c ="):
            c = list(map(float,
            stripped_line[4:].strip().strip('[]').split(',')))
        elif stripped_line.startswith("A ="):
            matrix_data = stripped_line[4:].strip()
            rows =
            matrix_data.strip().strip('[]').split('], [')
            for row in rows:
                A.append(list(map(float,
                row.strip().strip('[]').split(','))))
        elif stripped_line.startswith("b ="):
            b = list(map(float,
            stripped_line[4:].strip().strip('[]').split(', ')))

    c = np.array(c)
    A = np.array(A)
    b = np.array(b)

    return c, A, b
```

3.2 Estratégias de Implementação no Método Simplex

A implementação utiliza duas estratégias principais para otimizar o processo de seleção de pivôs e tratar a degenerescência:

- **Regra de Dantzig**: Para selecionar a variável de entrada na base, escolhe-se a variável com o coeficiente mais negativo na função objetivo.
- **Perturbação (Epsilon)**: Um pequeno valor, `EPSILON` (definido como $1e{-}18$), é adicionado para lidar com problemas de degenerescência e melhorar a estabilidade numérica, além de garantir com que se alcance o melhor valor ótimo na minimização, isto é, o valor mais negativo.

A implementação também inclui uma **Fase 1** para solucionar problemas infactíveis antes de entrar na fase principal do método simplex.

3.3 Classe Simplex e Métodos Principais

A seguir, apresenta-se a estrutura da classe `Simplex` e seus métodos principais:

```
class Simplex:
    def __init__(self, mode=MIN_MODE):
        self.main_variables_count = None
        self.restrictions_count = None
        self.objective_value = None
        self.basis = None
        self.mode = mode
        self.variables_count = 0
        self.iter = 0
        self.status = ""
```

Inicialização da Tabela Simplex: A tabela simplex é inicializada com as variáveis principais e restrições, e prepara a base inicial.

```
    def init_table(self, a, b):
        self.table = np.zeros((self.restrictions_count,
        self.variables_count + 1))
        for i in range(self.restrictions_count):
            for j in range(self.main_variables_count):
                self.table[i][j] = a[i][j]
```

```
            for j in range(self.restrictions_count):
                self.table[i][j +
                self.main_variables_count] = int(i == j)
            self.table[i][-1] = b[i]
```

Cálculo da Função Objetivo e Resolução: A função objetivo é calculada e o algoritmo segue iterativamente aplicando operações de Gauss para resolver o sistema.

```
    def calculate_f(self):
        self.f = -self.c.copy()
        for j, basis_var in enumerate(self.basis):
            self.f += self.c[basis_var] * self.table[j]
        self.objective_value = self.f[-1]
```

3.4 Fase 1 do Simplex

Para tratar problemas infactíveis, é implementada a Fase 1, onde uma solução básica inicial é criada para transitar para a Fase 2(que será chamada de simplex(b, A, c)):

```
def phase_1_simplex(self):
        THETA_INFINITE = -1
        opt = False
        n = len(self.table[0])
        m = len(self.table) - 2

        while not opt:
            min_cj = 0.0
            pivot_col = 1
            for j in range(1, n - m):
                cj = self.table[1][j]
                if cj < min_cj:
                    min_cj = cj
                    pivot_col = j
            if min_cj == 0.0:
                opt = True
                continue
```

```
            pivot_row = 0
            min_theta = THETA_INFINITE
            for i, xi in enumerate(self.table[2:], start=2):
                xij = xi[pivot_col]
                if xij > 0:
                    theta = xi[0] / xij
                    if theta < min_theta
                    or min_theta == THETA_INFINITE:
                        min_theta = theta
                        pivot_row = i

            if min_theta == THETA_INFINITE:
                self.status = "INFACTÍVEL"
                self.objective_value = None
                self.solution = None
                return 0

            self.table = self.pivot_on(
            self.table, pivot_row, pivot_col
            )
        return self.table
```

Solução e Otimização: A resolução da Fase 2 otimiza o problema conforme o modo selecionado (maximização ou minimização), que por padrão trata problemas de minimização conforme o contexto do trabalho. Basicamente função é executada até que se alcance uma solução ótima ou que o problema seja identificado como ilimitado ou infactível.

```
    def simplex(self, c: np.array, A: np.array, b: np.array):
        # com os novos casos de teste não é mais necessário
        # if self.mode == MIN_MODE:
        #     c = -c

        self.main_variables_count = A.shape[1]
        self.restrictions_count = A.shape[0]
        self.variables_count =
        self.main_variables_count + self.restrictions_count
        self.basis = [
        i + self.main_variables_count
        for i in range(self.restrictions_count)
```

```
]
self.c = np.concatenate(
[c, np.zeros((self.restrictions_count + 1))
]
)
self.f = np.zeros(
(self.variables_count + 1)
)
self.init_table(A + EPSILON, b + EPSILON)

if not self.remove_negative_b():
    self.status = 'INFACTÍVEL'
    return

self.table = self.phase_1_simplex()
if self.table.size == 0:
    self.objective_value = None
    return

while True:

    self.calculate_f()
    self.print_task()
    self.print_table()
    self.get_solve()

    #regra de dantzig
    candidate_columns = [
    i for i in range(len(self.f) - 1)
    if (self.f[i] < 0
    if self.mode == MAX_MODE
    else self.f[i] > 0)
    ]

    if not candidate_columns:
        self.status = 'ÓTIMO'
        break

    column = max(candidate_columns, key=lambda i: abs(self.f[i]))

    q = self.get_relations(column)
```

```
            if all(qi == np.inf for qi in q):
                self.status = 'INFINITO'
                self.objective_value = None
                self.solution = None
                break

            row = np.argmin(q)
            self.gauss(row, column)
            self.iter += 1
```

3.5 Visualização dos Resultados

Ao final de cada iteração, a tabela e o status são salvos em um arquivo para consulta e análise dos resultados do processo simplex, além disso, os dados requeridos, como status do problema i, valor da respectiva função objetivo, quantidade de iterações e a respectiva solução são mostrados 'printados' em quaisquer dos terminais de linha de comando.

```
def print_table(self):
     with open(os.path.join(
     'artifacts/output/', 'table.txt'), 'a') as f:
         f.write('     |' + ''.join(
         ['   y%-3d |' % (i + 1) for i in range(
         self.variables_count)]) + '    b   |\n')
         for i in range(self.restrictions_count):
             f.write('%4s |' % (
             'y' + str(self.basis[i] + 1)) +
             ''.join([' %6.2f |' % aij for j, aij
             in enumerate(self.table[i])]) + '\n')
         if self.status == "ÓTIMO":
             f.write('   F |' + ''.join(
             [' %6.2f |' % aij for aij in self.f]) + '\n')
             f.write('   y |' + ''.join(
             [' %6.2f |' % xi for xi
             in self.get_solve()]) + '\n\n')

 def print_coef(self, ai, i):
     if ai == 1:
         return 'y%d' % (i + 1)
     if ai == -1:
```

```
            return '-y%d' % (i + 1)
        return '%.2fy%d' % (ai, i + 1)

    def print_task(self):
        with open(os.path.join(
        'artifacts/output/', 'task.txt'), 'a') as f:
            f.write(' + '.join(['%.2fy%d' % (ci, i + 1)
            for i, ci in enumerate(
            self.c[:self.main_variables_count])
            if ci != 0]) + ' -> min\n')

            for i, a_row in enumerate(self.table):
                f.write(' + '.join([self.print_coef(ai, j)
                for j, ai in enumerate(
                a_row[:self.main_variables_count]) if ai != 0]) +
                ' = %.2f\n' % a_row[-1])
```

```
Problema 1

Status: ÓTIMO
Iterações: 5
Valor Ótimo: -35.69230769230769
Solução:
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          1.53846154  0.          0.          0.          0.
  0.          2.23076923  0.          0.          0.          0.
  0.          0.          2.07692308  0.          0.          0.
 12.07692308 11.23076923  0.          1.53846154  0.         16.15384615
 11.46153846  1.          0.         18.        ]

Problema 2

Status: INFINITO
Iterações: 10
Valor Ótimo: None
Solução: None

Problema 3

Status: INFACTÍVEL
```

```
Iterações: 0
Valor Ótimo: None
Solução: None
```

4 Discussão dos Resultados

Os resultados obtidos ao aplicar o algoritmo a três diferentes problemas de programação linear indicam um desempenho consistente e fidedignas aos resultados esperados, em relação às suas capacidades de identificação das soluções ótimas, bem como dos casos de solução ilimitada e inviável. A análise do **Problema 1** evidencia a capacidade do algoritmo em identificar soluções ótimas, caracterizadas pela convergência de um conjunto de valores que satisfazem todas as restrições impostas no modelo. Nesse caso, o algoritmo produziu um resultado que cumpre os limites especificados pelas restrições da matriz A e o vetor b, resultando em uma configuração de solução considerada "ÓTIMA". A solução ótima ocorre quando o algoritmo encontra um conjunto de valores x^* que satisfazem todas as restrições impostas pelo modelo. Neste caso, o algoritmo produziu um resultado que respeita os limites definidos pelas restrições da matriz A e pelo vetor b, resultando em uma solução considerada "ÓTIMA". No **Problema 2**, o algoritmo foi capaz de identificar corretamente a natureza infinita do problema. A solução ilimitada ocorre quando não há um limite superior para o valor da função objetivo, ou seja, a função objetivo pode ser aumentada indefinidamente sem violar as restrições do problema. A capacidade do algoritmo de detectar essa característica reflete sua robustez ao tratar problemas de natureza ilimitada, nos quais a viabilidade é mantida, mas a função objetivo pode ser maximizada sem limites definidos. Quando o programa detecta um problema dessa natureza, o status é dado como 'INFINITO'. Por fim, o **Problema 3** foi classificado como "INFACTÍVEL", indicando que nenhuma combinação de variáveis x satisfaz simultaneamente todas as restrições. A inviabilidade ocorre quando o conjunto de restrições forma um sistema contraditório, de modo que não existe uma solução viável que atenda a todas as condições. Esse resultado demonstra a capacidade do algoritmo em identificar a inviabilidade de uma solução dentro das condições fornecidas, reafirmando sua eficácia na detecção de inconsistências em problemas de programação linear. Esses resultados corroboram a precisão e a eficiência do algoritmo na análise de diferentes cenários de otimização linear, destacando sua capacidade de classificar problemas de acordo com as possibilidades de soluções viáveis, ilimitadas e inviáveis. Para efeito didático e de comparação os resultados,

no que se refere a iteração e valor da função objetivo, assemelham-se com resultados obtidos com a função linprog do scipy.optimize e a implementação do moonitor da disciplina.

5 Conclusão

Os testes realizados com três diferentes configurações de problemas de programação linear confirmam a eficácia do algoritmo proposto em resolver casos complexos e classificar corretamente os resultados de acordo com a natureza da solução. O algoritmo foi capaz de identificar corretamente uma solução "ÓTIMA", um caso de solução "ILIMITADA" e um caso "INFACTÍVEL". Tais resultados evidenciam a adequação do modelo implementado em lidar com problemas de otimização variados, oferecendo respostas precisas para cada tipo de problema, além de ser comprovadamente otimizado, no ponto em que alcança resultados similares à bibliotecas que contemplam a implementação do método Simplex, tal como scipy.optimize, além de corroborar os valores ótimos fornecidos pelo próprio monitor da disciplina. Além disso, a consistência com que o algoritmo abordou as restrições e a função objetivo de cada problema destaca sua robustez em contextos de programação linear com diferentes características. Este estudo fornece uma base sólida para futuras aplicações do algoritmo em cenários de otimização real, onde a capacidade de discriminar entre soluções viáveis, ilimitadas e inviáveis é de extrema importância para a tomada de decisão baseada em métodos de otimização.