Método simplex

Felipe Destaole 13686768
João Pedro Farjoun Silva 13731319
Luís Roberto Piva 13687727
Téo Sobrino Alves 12557192

Introdução

O algoritmo Simplex é uma técnica clássica usada para resolver problemas de otimização linear, que envolvem maximizar ou minimizar uma função linear sujeita a um conjunto de restrições lineares.

A implementação que fizemos está centrada em resolver problemas na forma padrão, através da Regra de Pivoteamento de Bland, utilizada em todo o processo em precisamos calcular o tableau dos problemas, e também no Simplex de Duas Fases, estudado em sala de aula.

Roteiro

-Descompactar arquivos .mps no formato Netlib;

-Implementação do tipos de leitura dos arquivos;

-Simplex de Duas Fases;

-Regra de Pivoteamento de Bland;

-Implementação do Simplex;

-Construção das tabelas resultados;

-Conclusão dos resultados.

Descompactar arquivos mps

- Ler arquivo em mps
- Separar as informações relevantes
  - Tipos das restrições
  - Função custo
  - Valor das restrições
- Transformar na forma padrão
  - Adicionar as restrições de variável
  - Adicionar as variáveis de folga
  - Separar variáveis negativas
- Retornar em formato txt
- Tipos de leitura

$$
\begin{array}{ccccc}
\multicolumn{2}{c}{\#\text{N}^{\text{o}} \text{ de restrições}} & & \multicolumn{2}{c}{\#\text{N}^{\text{o}} \text{ de variáveis}} \\
c_1 & c_2 & \dots & c_n & \\
A_{11} & A_{12} & \dots & A1n & b_1 \\
A_{21} & A_{22} & \dots & A2n & b_2 \\
 & & \vdots & & \\
A_{m1} & A_{m2} & \dots & A_{mn} & b_m
\end{array}
$$

Simplex de Duas Fases

O Simplex de Duas fases foi estudado em classe. O intuito desse método é, considerando um problema original na forma padrão, construímos um problema auxiliar adicionando variáveis artificiais, visando construir uma base "fácil", ou seja, uma matriz identidade para calcular um simplex com este problema auxiliar.

Na primeira fase, o foco está em encontrar uma solução básica para o problema original, utilizando-se dos dados e tableau do problema auxiliar. Essa solução é chamada de solução inicial. Aqui já podemos ver se o problema original tem solução, por exemplo.

A segunda fase, por sua vez, consiste na aplicação do simplex padrão, utilizando a solução inicial para otimizar a solução obtida na etapa anterior.

Pivoteamento de Bland

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.

Classe Simplex

```cpp
class Simplex
{
  private:
    MatrixXf T;  // tableau
    MatrixXf A;  // restrictions
    MatrixXf B;  // base
    ArrayXf b;   // restriction costs
    ArrayXf c;   // objective function
    ArrayXf X;   // variables in the base (basic variables)
    ArrayXf c_r; // reduced costs
    double EPS;
    int iter_num;
    int f_iter_n;
    int s_iter_n;

    int restrictions;
    int variables;

    ArrayXf SolveTableau(MatrixXf &A, ArrayXf &c, ArrayXf &X);
    MatrixXf BuildTableau(ArrayXf cst);
    void UpdateTableau();

  public:
    Simplex();
    Simplex solve();
    ArrayXXf SolveFirstPhase();
    ArrayXXf SolveSecondPhase();
    You, 4 days ago | 1 author (You)
    class Reader
    {
      private:
        std::fstream file_d;

      public:
        Simplex Read();
        Simplex ReadMatrix(const std::string path);
        void ReadLineFiletoVec(std::vector<float> &vec);
    };
};
```

Método Solve

```
Simplex Simplex::solve()
{

    Reader r = Reader();
    Simplex s = r.Read();

    double delta_t = 0;
    std::vector<double> timer;

    for (int i = 0; i < REPS; i++) {
        auto t_0 = std::chrono::high_resolution_clock::now();
        s.SolveFirstPhase();
        s.f_iter_n = s.iter_num;
        s.SolveSecondPhase();
        s.s_iter_n = s.iter_num - s.f_iter_n;
        auto t_1 = std::chrono::high_resolution_clock::now();
        delta_t =
            std::chrono::duration_cast<std::chrono::nanoseconds>(t_1 - t_0)
                .count();
        delta_t *= 1e-9;
        timer.push_back(delta_t);
    }
```

Primeira Fase

```cpp
ArrayXXf Simplex::SolveFirstPhase()
{
    // adapt the problem to the first phase

    // change cost function
    ArrayXf cst(variables + restrictions + 1);

    // cost function is 1 in artificial variables and 0 elsewhere
    cst = ArrayXf::Zero(cst.size());
    cst.tail(restrictions) = 1;

    // build the tableau
    T = BuildTableau(cst);

    // get the initial base (artifical variables for the first phase)
    float *data = (float *)malloc(restrictions * sizeof(float));
    for (int i = variables; i < variables + restrictions; i++) {
        data[i - variables] = i + 1;
    }
    Eigen::Map<ArrayXf> base(data, restrictions);
    ArrayXf bb = base;

    X = SolveTableau(T, cst, bb);
    UpdateTableau();

    free(data);

    return X;
}
```

Resolvendo o Tableau

```cpp
ArrayXf Simplex::SolveTableau(MatrixXf &T, ArrayXf &c, ArrayXf &X)
{
    // calculates reduced cost based on cost array
    ArrayXXf cr = ReducedCost(T, c, X);
    // update reduced cost in the tableau
    T.topRows(1) = cr.transpose();

    double max = T.maxCoeff();
    double min = find_smallest_abs(T);

    //define float opeartion error threshold
    EPS = std::max(1E-5, min/(max*max));

    int idx_enter_base = 0;
    int idx_remove_base = 0;
    while (idx_enter_base != -1) {
        iter_num++;
        // apply bland's rule
        idx_enter_base = find_var_add_base(T.row(0), EPS);
        idx_remove_base = find_var_remove_base(T, idx_enter_base, EPS);

        // if there is a negative reduced cost and no available variable to exit
        // the base, the problem is unbounded
        if (idx_remove_base == -1 && idx_enter_base != -1) {
            throw std::domain_error(
                "Invalid restrictions (unbounded problem)\n");
            exit(1);
        }
        insert_in_base(T, X, idx_enter_base, idx_remove_base);
    }

    return X;
}
```

Atualizando o Tableau

```cpp
void Simplex::UpdateTableau()
{
    MatrixXf updated_tableau(restrictions + 1, variables + 1);
    updated_tableau = T.block(0, 0, restrictions + 1, variables + 1);

    // remove artifical variables from last columns and from base (if any)
    for (int i = 1; i <= X.size(); i++) {
        if (X[i - 1] > variables) {
            remove_row(updated_tableau, i);
            X = remove_item(X, i - 1);
        }
    }

    // checks if the solution is feasible
    auto R = Eigen::FullPivLU<MatrixXf>(A);
    if (updated_tableau.col(0).size()-1 < R.rank()) {
        throw std::domain_error("Invalid restrictions (infeasible solution)\n");
        exit(1);
    }

    T = updated_tableau;
}       You, last week • fixed reduced cost
```

Segunda Fase

```
ArrayXXf Simplex::SolveSecondPhase()
{
    // update cost function to recalculate reduced costs
    ArrayXf cst = ArrayXf::Zero(variables + 1);
    cst.tail(variables) = c;
    X = SolveTableau(T, cst, X);

    return X;
}
```

Resultados

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

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

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

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

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

Resultados

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

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

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

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

Conclusão

1. Precisão da função objetivo

Em comparação com os resultados dispostos na biblioteca Netlib, os erros médios cometidos pela implementação foram consistentes e precisos, geralmente mantendo-se abaixo de 10%.

Instâncias de grande porte e alta complexidade tendem a demandar um maior número de iterações, o que contribuiu para uma ligeira elevação nos erros médios de tais instâncias, como podemos perceber nas tabelas.

Assim, a variação nos erros médios foi influenciada pelo tamanho da instância e sua complexidade.

Conclusão

2. Números de iterações

O número de iterações está relacionado à complexidade e ao tamanho da instância considerada. Um exemplo de instância que aconteceu isso foi SCAGR25.

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.

Conclusão

3. Sensibilidade do Parâmetro epsilon

O parâmetro epsilon regula a tolerância de erro nas operações de ponto flutuante. Escolhemos trabalhar com ponto flutuante para minimizar o uso de memória.

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.

Observamos que a utilização de epsilon menor em problemas mais extensos e complexos poderia levar o algoritmo a classificar o problema como "unbounded" (ilimitado). Por isso, é importante o calibre inteligente desse parâmetro.

Conclusão

4. Tempo de Execução

A relação entre o tempo de execução do algoritmo simplex, o tamanho e a complexidade das instâncias é diretamente proporcional.

Problemas de maior envergadura e complexidade demandaram um maior número de iterações, resultando em um tempo de execução proporcionalmente mais extenso.

Os tempos médios de execução se mostraram relativamente curtos para a maioria das instâncias testadas. Mesmo em cenários mais desafiadores, exemplificados pelas instâncias "ISRAEL" e "SCAGR25", os tempos de execução se mantiveram razoáveis.

OBRIGADO!