Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

# Ferramenta de CAD para o Desenvolvimento do Layout de Capacitores em Circuitos Integrados CMOS

Autor:

_____

Nilson Carvalho Silva Junior

Orientador:

_____

Prof. Carlos Fernando Teodósio Soares, Ph. D.

Examinador:

_____

Prof. Antonio Petraglia, Ph. D.

Examinador:

_____

Prof. José Gabriel Rodriguez Carneiro Gomes, Ph. D.

DEL

Março de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

## AGRADECIMENTO

Agradeço a todas as pessoas que possibilitaram a conclusão deste trabalho. Agradeço, primeiramente, à minha família pela confiança depositada em mim e pelo suporte dado, em especial à minha mãe Magaly.

Aos meus professores, em especial, ao meu orientador Carlos Fernando Teodósio, pela disponibilidade e paciência para me ensinar e tirar dúvidas.

Aos meus amigos, em especial, Bernardo Cid Killer, com quem desenvolvi o projeto, e Alan Endalécio e Vitor Borges, com quem fiz diversos trabalhos durante a graduação.

Ao CNPq, pelas bolsas pagas.

Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

## RESUMO

Esta dissertação apresenta o desenvolvimento de uma ferramenta de CAD (*Computer-Aided Design*), cuja função principal é organizar, automaticamente, capacitores unitários em uma dada matriz, de modo a minimizar as interferências do processo de fabricação de circuitos integrados CMOS.

O algoritmo de otimização utilizado foi o *Simulated Annealing*. Esse algoritmo foi escolhido por ser bastante adequado a problemas discretos de minimização. O presente projeto visa automatizar o processo de desenvolvimento do *layout* de Circuitos a Capacitores Chaveados.



Palavras-Chave: CAD, Capacitores Chaveados, Circuitos Integrados, CMOS, Simulated Annealing.

**ABSTRACT**

This dissertation presents the development of a Computer-Aided Design tool, whose main function is automatically organize unit capacitors in a given matrix, in order to minimize interferences in CMOS integrated circuits fabrication process.

The Simulated Annealing is the optimization adopted in this work. This algorithm is adopted because it is very suitable to discrete minimization problems. This project aims to automate the process of developing the layout of Switched Capacitors Circuit.

Key-words: Computer-Aided Design, Switched Capacitors, Integrated Circuits, CMOS, Simulated Annealing.

## SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

CAD – *Computer-Aided Design*

CMOS – Complementary Metal-Oxide Semiconductor

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

## 1.1 – Tema

O tema do trabalho é o descasamento de componentes em circuitos integrados analógicos CMOS. Neste sentido, pretende-se propor maneiras de contornar ou minimizar esse problema.

## 1.2 – Delimitação

O objeto de estudo são os métodos para criação de *layouts* de modo a minimizar os efeitos dos erros de descasamento dos capacitores em Circuitos a Capacitores Chaveados. Erros estes que podem ser classificados em sistemáticos e aleatórios, e acontecem devido a variações nos parâmetros do processo de fabricação CMOS e a adição de elementos parasitas. O projeto se limitará à compensação dos efeitos dos gradientes de variação de processo e à minimização da variância entre os capacitores dos filtros, de modo a melhorar o casamento entre eles. Além disso, o processo de criação do *layout* será automatizado, uma vez que nem sempre é trivial encontrar a alocação ótima dos componentes manualmente.

## 1.3 – Justificativa

Como será mostrado no Capítulo 2; em Filtros a Capacitores Chaveados, todos os coeficientes dependem dos valores relativos entre capacitâncias. Sendo assim, descasamentos entre capacitores se tornam bastante problemáticos, pois alteram os coeficientes das funções de transferência dos circuitos, introduzindo erros na resposta em frequência dos filtros. No presente trabalho, a fim de minimizar os efeitos do descasamento, serão utilizados capacitores unitários de mesmo valor na elaboração do

*layout* dos filtros, já que estes podem ser fabricados de forma bem casada. Desse modo, serão feitas associações em paralelo de capacitores unitários quando necessitarmos de capacitores de valores maiores no circuito. Assim, as capacitâncias realizáveis no projeto deverão ser múltiplos inteiros da capacitância de um único capacitor unitário.

Esses capacitores unitários serão dispostos na forma de matrizes no *layout* do circuito integrado. Porém, encontrar a solução ótima de alocação dos capacitores unitários na matriz, minimizando os erros de descasamento, é uma tarefa demasiadamente complexa para ser realizada manualmente, em especial nos casos em que o número de capacitores a serem casados é grande. Assim, o trabalho apresenta uma proposta para a melhoria dos métodos utilizados no projeto de CI's e resolve o problema através do desenvolvimento de uma ferramenta de CAD (*Computer-Aided Design*).

## 1.4 – Objetivos

O objetivo geral é, então, desenvolver uma ferramenta computacional que auxilie e automatize parte do processo de projeto do *layout* de circuitos analógicos integrados, com ênfase em circuitos que necessitem do casamento entre capacitores, como no caso de filtros a capacitores chaveados. Mais especificamente, a ferramenta permitirá que o usuário especifique as dimensões da matriz e o número de capacitores unitários em paralelo empregados na construção de cada capacitor do circuito, além da configuração dos parâmetros do algoritmo de otimização responsável pela busca do *layout* ótimo. De posse desses dados, o software será capaz de encontrar a solução para a alocação dos componentes, minimizando o descasamento e maximizando o coeficiente de correlação entre os componentes que necessitam estar casados.

## 1.5 – Metodologia

Este trabalho irá utilizar o algoritmo *Simulated Annealing* para encontrar as matrizes ótimas para os *layouts* dos capacitores unitários. A função-custo do algoritmo envolve o cálculo de parâmetros de desempenho relacionados com o casamento entre os

capacitores presentes nos circuitos. O algoritmo é, em geral, utilizado para resolver problemas de minimização de natureza discreta, como o problema do presente trabalho.

Seu funcionamento mimetiza o resfriamento de um sistema. Em estruturas cristalinas, o arranjo atômico se altera para uma configuração de baixa agitação térmica e, portanto, de baixa energia quando resfriado. Definindo uma nova função-custo no lugar da energia e simulando um resfriamento por meio de um algoritmo, é possível encontrar mínimos de funções.

É importante ressaltar que, a cada passo, uma nova solução tem determinada chance de ser aceita, de acordo com a distribuição de probabilidades de Boltzmann, e essa chance é decrescente com a temperatura. Logo, o algoritmo é probabilístico e não é certo que encontre a solução ótima para todos os casos. Além disso, a cada execução do algoritmo, uma diferente solução pode ser obtida. Alterando-se, porém, seus parâmetros, é possível aumentar a chance de encontrar a melhor solução, bem como alterar o tempo de execução do mesmo.

## 1.6 – Descrição

Abaixo será apresentado uma prévia do que será visto nos próximos seis capítulos.

O Capítulo 2 apresenta algumas características interessantes dos Filtros a Capacitores Chaveados e a dependência desses filtros com respeito ao casamento entre os capacitores do circuito. Exemplos de dois circuitos integradores implementados com Capacitores Chaveados são apresentados, assim como suas funções de transferência. Por fim, são apresentadas as soluções mais conhecidas para os problemas de descasamento entre os capacitores do filtro.

Como a solução proposta no Capítulo 2 para o problema do descasamento entre capacitores pode ser obtida por meio de um algoritmo de otimização, o Capítulo 3, primeiramente, apresenta uma função-custo inicialmente proposta, depois, são apresentados os cálculos das variáveis utilizadas nessa função – o erro de centroide comum e os coeficientes de correlação entre dois capacitores. O funcionamento do algoritmo, de modo básico é apresentado e, por fim, uma nova função-custo é proposta – esta última foi utilizada no programa.

A plataforma de desenvolvimento e as classes que compõem o programa são apresentados no capítulo 4. Além disso, nele é explicitado a função dos parâmetros.

No Capítulo 5, é mostrado passo-a-passo a utilização do programa, além disso, são mostrados alguns resultados de *layouts* gerados com ele.

O último capítulo é reservado para a conclusão e algumas propostas de trabalho futuro, com a finalidade tornar o *software* desenvolvido mais funcional.

# Capítulo 2

# Filtros a Capacitores Chaveados

## 2.1 – Fundamentação Teórica

Filtros são estruturas que processam sinais aplicados à sua entrada e apresentam na saída uma resposta composta por um subconjunto dos componentes do sinal de entrada com variações de fase e amplitude dependentes da frequência. Os filtros podem ser divididos em duas categorias: analógicos e digitais. Este trabalho se delimitará aos analógicos.

A principal diferença entre os filtros analógicos e digitais é que, nos filtros digitais, os sinais de entrada e de saída podem assumir apenas uma quantidade finita de valores, de modo a serem representados por palavras binárias. Por outro lado, nos filtros analógicos, tanto os sinais de entrada quanto os sinais de saída podem assumir uma quantidade infinita de valores.

Os filtros também podem ser classificados como discretos ou contínuos no tempo, de acordo com o tipo de sinais que eles processam. Para um filtro contínuo, é possível definir os valores dos sinais de entrada e de saída em qualquer instante de tempo. A Figura 2.1 mostra um exemplo de sinal contínuo no tempo.

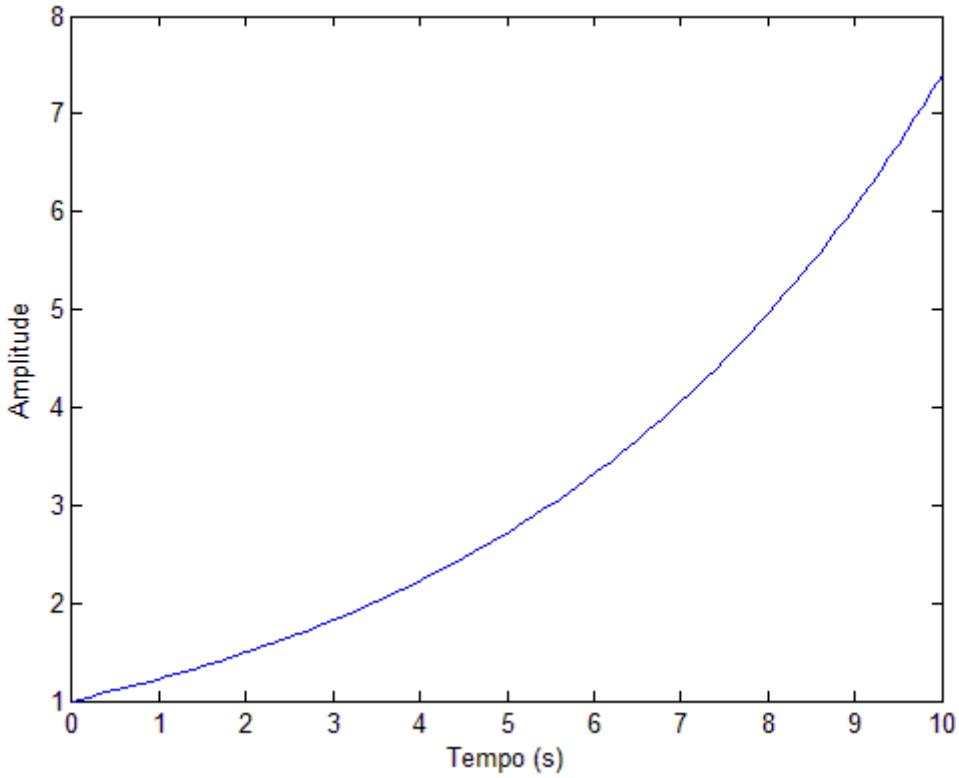


Figura 2.1 – Sinal Contínuo no tempo.

Os sinais processados por filtros discretos no tempo, por outro lado, não apresentam valores de entrada e saída definidos para todos os instantes de tempo, mas apenas em instantes discretos que são múltiplos do período de amostragem. A Figura 2.2 apresenta um exemplo no qual é feita uma amostragem a cada segundo.

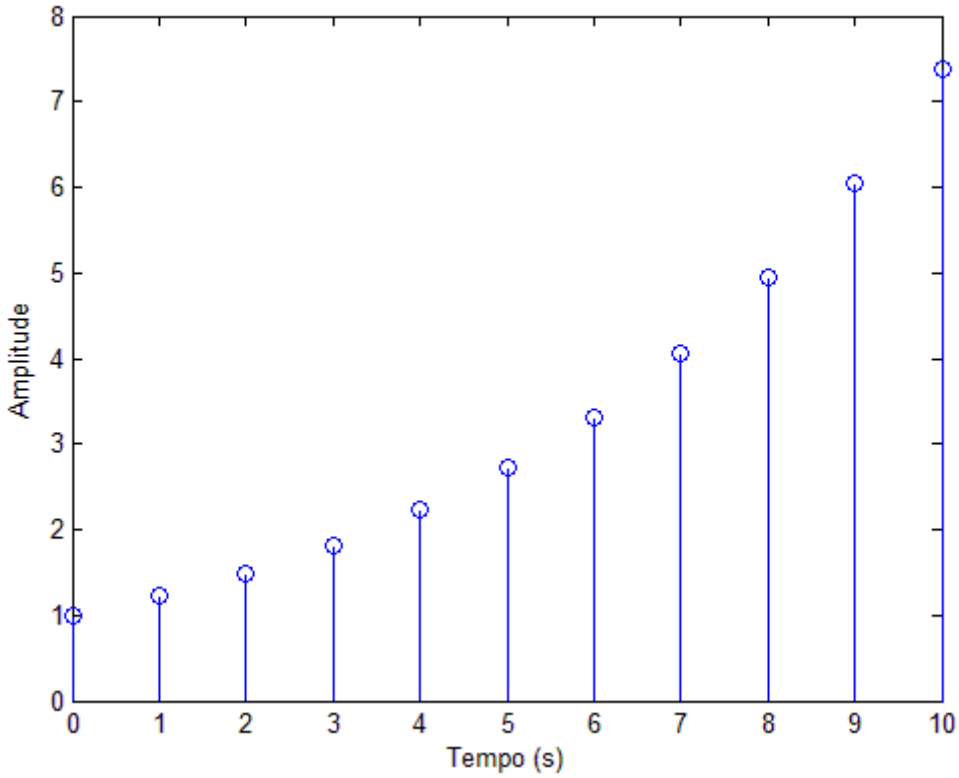


Figura 2.2 – Sinal Discreto no tempo.

Os Filtros a Capacitores Chaveados são discretos e analógicos, porém processam sinais a dados amostrados, ou seja, os sinais são contínuos no tempo, mas só podem alterar seu valor em instantes discretos de tempo, mantendo seu valor constante por um intervalo de tempo fixo T entre esses instantes. Sinais assim podem ser obtidos passando sinais contínuos por um circuito do tipo *sample-and-hold*. A Figura 2.3, apresenta um sinal amostrado e retido (*sampled-and-held*).

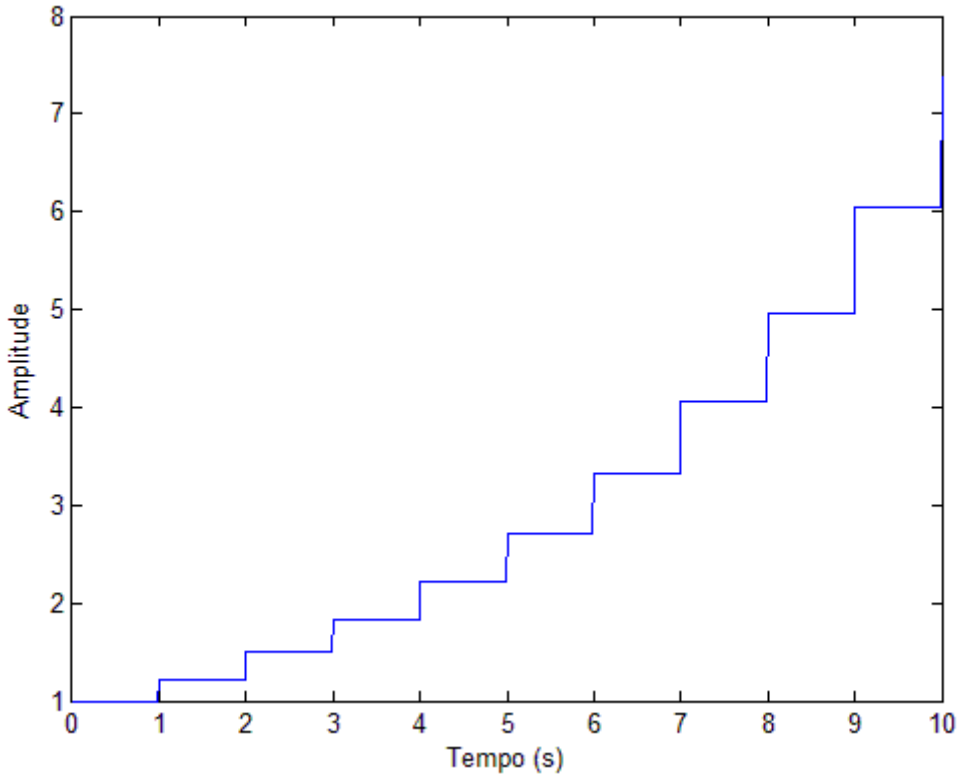


Figura 2.3 – Sinal *sampled-and-held*.

## 2.2 – Capacitores Chaveados

A Figura 2.4 a seguir, explica como simular um resistor com um capacitor chaveado.

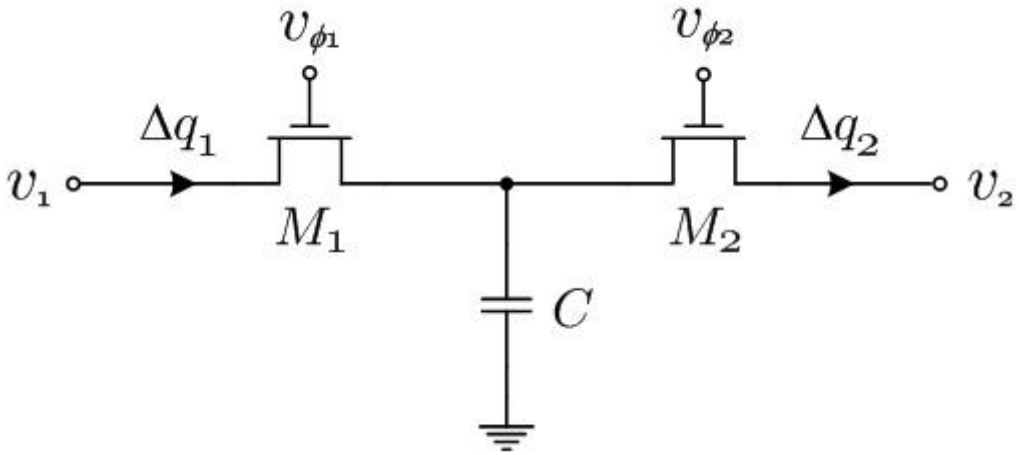


Figura 2.4 – Exemplo de um resistor implementado através de um capacitor chaveado.

Os sinais de controle das chaves $M_1$ e $M_2$ apresentam uma diferença de fase de 180°. Quando $M_1$ encontra-se fechada e $M_2$ aberta, a tensão sobre o capacitor C passa a ser $V_1$. Quando, então, $M_2$ fecha e $M_1$ abre, a tensão entre os terminais de C passa a ser $V_2$. A variação de carga armazenada no capacitor é, então

$$\Delta q = C(V_1 - V_2) \tag{II.1}$$

Logo, a corrente média que flui pelo capacitor em um período de chaveamento é

$$i = \frac{\Delta q}{T} = \frac{C(V_1 - V_2)}{T} \tag{II.2}$$

Sendo $T$ o período de chaveamento de $M_1$ e $M_2$. Porém, é sabido que, pela Lei de Ohm, a diferença de potencial entre dois pontos 1 e 2 é dada por:

$$V_1 - V_2 = R * i \tag{II.3}$$

Onde, $R$ representa a resistência elétrica e $i$ representa a corrente elétrica. Comparando as equações (II.2) e (II.3), é possível perceber que uma resistência de valor $R = T/C$ pode ser implementada por meio de um Capacitor Chaveado de capacitância $C$, utilizando um período de chaveamento $T$.

A implementação de resistores dessa forma apresenta grandes vantagens em relação à implementação direta de resistores. Uma delas é quanto ao tamanho: resistores grandes ocupam uma grande área de silício policristalino, porém, podem ser mais facilmente implementados com esse tipo de circuito, já que diminuindo o período de chaveamento, é possível aumentar sua resistência equivalente. Para efeito de comparação, um resistor de 10 MΩ poderia ser implementado com uma frequência de chaveamento de 200 kHz e capacitores de 0.5 pF em uma área de 0.01 mm², enquanto um resistor de mesmo valor, com tecnologia CMOS utilizaria uma área próxima a 1 mm² [8]. Além disso, resistores apresentam grandes variações em seus valores com a temperatura, além de uma pobre linearidade [7]. Por último, a precisão poderia ser citada, como será visto mais à frente, é possível fazer filtros precisos utilizando Capacitores Chaveados mesmo quando os componentes apresentam elevadas variações nos seus valores absolutos de capacitância.

## 2.3 – Integradores Forward e Backward de Euler

A seguir, será explicado o funcionamento de dois circuitos a capacitores chaveados muito úteis na construção de filtros: os integradores. Primeiramente, o Integrador Forward de Euler é apresentado na Figura 2.5.

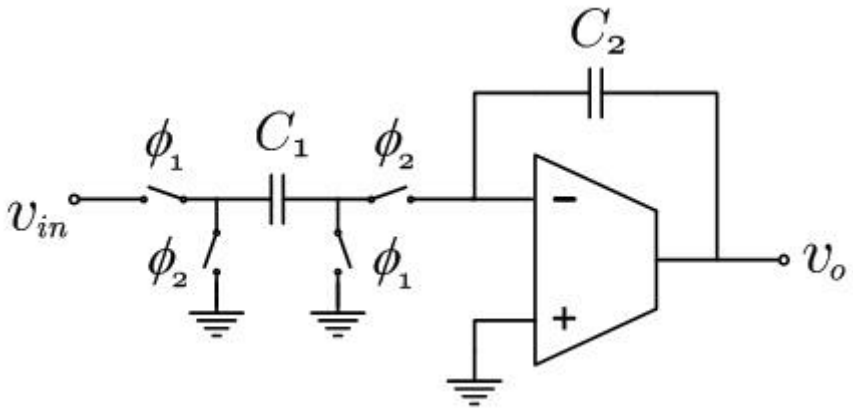


Figura 2.5 – Integrador Forward de Euler.

As chaves $\emptyset_1$ e $\emptyset_2$ têm uma diferença de fase de 180°, ou seja, quando as chaves $\emptyset_1$ estão abertas, as chaves $\emptyset_2$ estão fechadas e vice-versa. Inicialmente, será suposto $\emptyset_1$ fechada e $\emptyset_2$ aberta. Assim, $V_o = V_0(t)$ e $V_{in} = V_{in}(t, )$.

Após meio período, as chaves $\emptyset_1$ abrem e as chaves $\emptyset_2$ fecham. Assim, a carga armazenada em $C_1$ flui para $C_2$. Logo, a carga $q_2$ armazenada em $C_2$ pode ser calculada como $q_2(t+T) = q_2(t) + q_1(t)$:

Substituindo $q = C * V$,

$$C_2 * V_o(t+T) = C_2 * V_o(t) + C_1 * V_{in}(t), \quad \text{(II.4)}$$

Utilizando a Transformada Z:

$$C_2 * z * V_o(z) = C_2 * V_o(z) + C_1 * V_{in}(z), \quad \text{(II.5)}$$

$$C_2 * V_o(z) * (z-1) = C_1 * V_{in}(z), \quad \text{(II.6)}$$

$$\frac{V_o(z)}{V_{in}(z)} = \frac{C_1}{C_2 * (z-1)} = \frac{C_1 * z^{-1}}{C_2 * (1-z^{-1})}, \quad \text{(II.7)}$$

Esse circuito, então, aproxima a função integral por $z^{-1}/(1-z^{-1})$. Ou seja,

$$\int_{(n-1)T}^{nT} f(t)dt = T * f((n-1)T) \quad \text{(II.8)}$$

que corresponde à aproximação numérica Forward de Euler.

A Figura 2.6 mostra o Integrador Backward de Euler a capacitores chaveados.

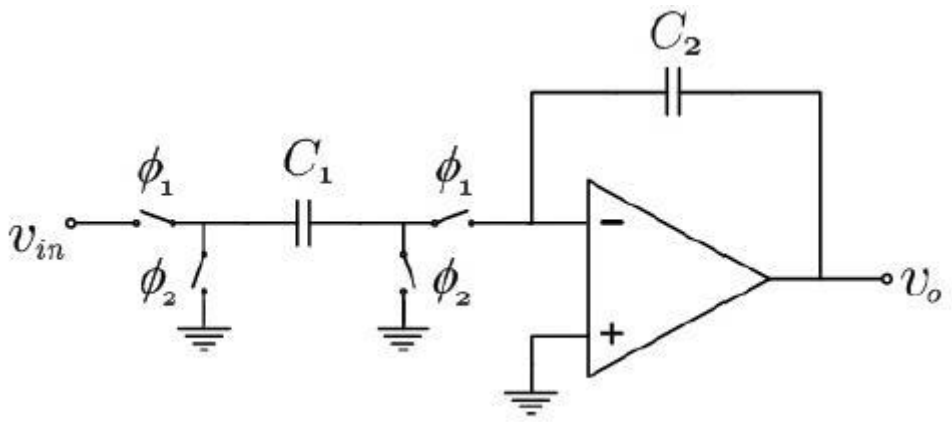


Figura 2.6 – Integrador Backward de Euler.

Quando a chave $\emptyset_1$ fecha, $C_1$ adquire a tensão $V_{in}(t)$. Após meio período, $\emptyset_1$ abre e $\emptyset_2$ fecha. O capacitor $C_2$, então, adquire a carga de $C_1$. Após mais meio período, o ciclo recomeça. Logo:

$$q_2(t+T) = q_2(t) + q_1(t+T), \tag{II.9}$$

Substituindo as cargas:

$$C_2 * V_o(t+T) = C_2 * V_o(t) + C_1 * V_{in}(t+T), \tag{II.10}$$

Ou seja:

$$\frac{V_o(z)}{V_{in}(z)} = \frac{C_1}{C_2} * \frac{1}{(1-z^{-1})}, \tag{II.11}$$

Esse circuito aproxima a integral do sinal de entrada como:

$$\int_{(n-1)T}^{nT} f(t)dt = T * f(nT) \tag{II.12}$$

que corresponde à aproximação numérica Backward de Euler.

Nos exemplos acima, é possível perceber que as funções de transferência dos integradores dependem apenas de razões entre as capacitâncias de $C_1$ e $C_2$. Essa é uma característica comum a todos os Filtros a Capacitores Chaveados [9], onde os coeficientes dos polinômios das suas funções de transferência dependem exclusivamente de razões de capacitâncias.

## 2.4 – Erros em Circuitos Integrados

Erros absolutos de capacitores em Circuitos Integrados CMOS são, em geral, da ordem de 20% a 40% [1] de sua capacitância. Porém, apesar do erro nos valores absolutos, a razão entre as capacitâncias de dois capacitores de mesmo valor nominal pode ser bem pequena, chegando a 0.1% [2], caso sejam utilizadas técnicas adequadas de layout. Portanto, seria interessante projetar filtros nos quais os coeficientes dos polinômios da função de transferência dependessem dos valores relativos entre os componentes, em vez de seus valores absolutos. Os Filtros a Capacitores Chaveados, conforme mencionado acima, apresentam essa característica. Portanto, é possível implementá-los em circuitos integrados CMOS com pequenos desvios nos coeficientes da função de transferência em relação aos valores nominais projetados.

Além da precisão, esses filtros são implementados utilizando apenas amplificadores operacionais, capacitores e chaves analógicas. Isso os torna ainda mais atrativos, pois resistores e indutores são, em geral, de difícil implementação e funcionam de maneira pouco linear quando implementados com tecnologia CMOS.

## 2.5 – Erros de Descasamento de Capacitores

Pode haver erros de descasamento de variadas origens quando projetamos um Circuito a Capacitores Chaveados. Estes erros podem ser divididos em sistemáticos e aleatórios.

Os Erros Aleatórios são, em geral, provocados por variações na espessura no óxido do *chip* e irregularidades nas bordas dos capacitores (que alteram sua área e seu perímetro, e, consequentemente, sua capacitância). Tais erros não podem ser eliminados pelo projetista, apenas atenuados.

Os Erros Sistemáticos são provocados, frequentemente, pelas capacitâncias parasitas que surgem quando implementamos o *layout* em *chips* e pelo gradiente de variação de processo. Tais erros poder ser eliminados através do uso de práticas adequadas de layout.

Apesar das vantagens citadas na Sessão 2.4, o valor dos capacitores unitários, quando implementados na matriz, varia de acordo com a posição em virtude do gradiente de processo. A espessura do óxido e outros parâmetros do *chip* variam de maneira aproximadamente linear com a posição no *layout*. Essa variação se reflete na

capacitância, fazendo com que ela também varie de maneira aproximadamente linear, obedecendo à seguinte equação [3]:

$$C = C_o + \alpha(x - x_o) + \beta(y - y_o)\,, \tag{II.13}$$

sendo $C_o$ o valor assumido pelo capacitor na posição $(x_o, y_o)$ da matriz do *chip*, $(x, y)$ a posição do capacitor na matriz e α e β constantes dependentes do gradiente de variação de processo.

É possível minimizar as interferências dos parâmetros do gradiente de processo mesmo sem conhecer as constantes α e β, utilizando técnicas que serão apresentadas mais à frente.

## 2.6 – Técnicas para Casamento

Os erros acima podem ser minimizados utilizando algumas técnicas. A primeira a ser apresentada é o emprego de Capacitores Unitários. Esta técnica consiste em utilizar todos os capacitores com a mesma área, o mesmo perímetro e, teoricamente, mesmo valor de capacitância no circuito, desconsiderando-se as variações nos parâmetros do processo de fabricação. Associações em paralelo devem ser utilizadas para implementar capacitores de maiores valores. Desse modo, o erro relativo entre os capacitores será menor, já que as variações nos parâmetros de processo tendem a afetar os capacitores unitários de maneira semelhante se o *layout* do CI for cuidadosamente elaborado.

Assim, quando desejarmos implementar um Filtro a Capacitores Chaveados, primeiramente será necessário aproximar seus coeficientes por razões de inteiros. A partir disso, serão utilizados capacitores unitários para implementá-los. Por exemplo, caso $C_1/C_2 \cong 2/3$ em um determinado circuito, deverão ser utilizados 2 Capacitores Unitários em paralelo para $C_1$ e 3 Capacitores Unitários em Paralelo para $C_2$. Do mesmo modo, para um circuito que apresenta $(C_1 + C_2)/C_3 \cong 7/3$ e $C_3/C_2 \cong 4/3$, deverão ser utilizados 19 Capacitores Unitários em paralelo para $C_3$, 9 Capacitores Unitários em paralelo para $C_2$ e 12 Capacitores Unitários em paralelo para $C_3$.

Como o arranjo dos capacitores será matricial, após escolher o número de linhas e colunas da matriz de capacitores unitários, pode acontecer de o número de capacitores utilizados não ser suficiente para completar a matriz. No primeiro exemplo de matriz acima, em que $C_1/C_2 \cong 2/3$, seriam utilizados 5 capacitores unitários. Se fosse utilizada uma matriz contendo 2 linhas e 3 colunas, sobraria um espaço na matriz.

Nestes casos, são utilizados capacitores *dummy*, que completam os espaços na matriz, fazendo com que todos os capacitores vejam as mesmas fronteiras, evitando variações de capacitância devido ao efeito de franja de campo elétrico.

Os capacitores unitários resolvem uma parte dos problemas apresentados. O gradiente de variação de processo ainda precisa ser considerado e mitigado. Para isso, são utilizadas geometrias de centroide comum.

A geometria de centroide comum é definida da seguinte forma: visualizando cada capacitor como uma massa, o centro da matriz deve coincidir com o centro de massa (centroide) de cada capacitor. A Figura 2.7 mostra uma estrutura dessa forma.

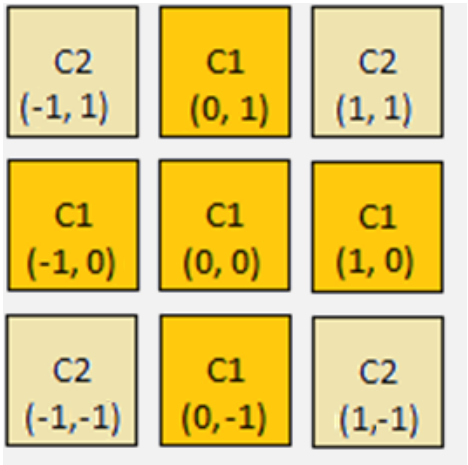


Figura 2.7 – Estrutura de centroide comum.

Esse tipo de estrutura anula os erros gerados pelo Gradiente de Variação de Processo. Utilizando o centro da matriz como referência para as coordenadas x e y, o valor de cada capacitância unitária, em função da posição, seria dado por $C \cong C_{0,0} + \alpha * x + \beta * y$. Ou seja, para o capacitor $C_k$, composto por $n$ unitários $C_i$:

$$C_k = \sum_{i=1}^{n} C_i \tag{II.14}$$

$$C_k = \sum_{i=1}^{n} C_{0,0} + \alpha * x_i + \beta * y_i \tag{II.15}$$

$$C_k = n\, C_{0,0} + \left(\alpha \sum_{i=1}^{n} x_i\right) + \left(\beta \sum_{i=1}^{n} y_i\right), \tag{II.16}$$

porém, para cada capacitor, o centro de massa coincidiria com o centro da matriz, logo, $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = 0$. Ou seja, os parâmetros do gradiente de processo $\alpha$ e $\beta$, apesar de interferirem individualmente no valor dos unitários, não interferem mais no valor das capacitâncias $C_k$ formada pela associação deles em paralelo, quando essa geometria é utilizada. Isto acontece para toda geometria de centroide comum.

É importante ressaltar que nem sempre é possível obter estruturas em centroide comum. Se fossem necessários, por exemplo, três capacitores unitários para $C_1$ e um capacitor unitário para $C_2$, não seria possível obter uma estrutura em centroide comum utilizando uma matriz de duas linhas e duas colunas.

O próximo passo seria tentar mitigar os erros aleatórios existentes nos circuitos. Para isso, poderia ser utilizada a Regra de Pelgrom [5]:

$$\sigma_C^2 = \frac{{A_p}^2}{W * L} + {S_p}^2 * D^2, \tag{II.17}$$

onde $A_p$ e $S_p$ são parâmetros do processo de fabricação, $W$ e $L$ representam, respectivamente, a largura e o comprimento dos capacitores unitários analisados, $D$ a distância entre eles e $\sigma_C^2$ seu momento central de segunda ordem dado por $E[C - \bar{C})^2]$, onde $E$ denota a operação valor esperado, ou seja, a variância observada. Segundo essa regra, a variância de um capacitor é inversamente proporcional às suas dimensões, além disso, a mesma aumenta com a distância entre os capacitores. Para garantir um bom casamento de capacitores, uma boa prática seria calcular as variâncias de cada capacitor. A configuração que apresentasse a menor variância máxima seria a escolhida.

Assim, se os capacitores unitários forem considerados como variáveis aleatórias independentes e identicamente distribuídas, um capacitor formado por $N$ unitários apresentaria:

$$\frac{{\sigma_C}^2}{E[C]} = \frac{{\sigma_{Cu}}^2}{N * Cu}, \tag{II.18}$$

onde, ${\sigma_{Cu}}^2$ seria a variância de um capacitor unitário e $Cu$ seu valor médio. Deste modo, a variância seria tanto maior quanto menor o número de unitários que formam um capacitor, e, obedecendo a Regra de Pelgrom, a melhor matriz solução seria a que apresentasse maior espalhamento para os capacitores com o maior número de unitários e vice-versa. Porém, é possível considerar um modelo mais realista e complexo que esse.

De acordo com [4], é possível diminuir as flutuações aleatórias dos componentes em um chip quando se considera a correlação entre os componentes.

Suponha que haja dois capacitores no circuito a ser projetado: $C_s$, formado pela associação em paralelo dos unitários $C_{s1}, C_{s2}, \dots, C_{sp}$, e $C_t$, formado pela associação em paralelo dos $q$ unitários $C_{t1}, C_{t2}, \dots, C_{tq}$. O objetivo do programa estará cumprido caso a configuração matricial seja a que minimize a variância $\sigma^2(C_s/C_t)$ (ou $\sigma^2(C_t/C_s)$), dentre as configurações que apresentam um desvio mínimo em relação ao arranjo com centroide comum. Como todos os capacitores unitários são construídos da mesma forma, pode ser considerado que todos os unitários têm a mesma variância $\sigma^2(C_{un})$. Sendo $\sigma^2(C_{si}, C_{tj})$ a covariância entre os capacitores $C_{si}$ e $C_{tj}$, o coeficiente de correlação entre eles é dado por:

$$\rho(C_s, C_t) = \frac{\sigma^2(C_s, C_t)}{\sigma(C_s)\sigma(C_t)}, \tag{II.19}$$

logo,

$$\rho(C_s, C_t)\sigma(C_s)\sigma(C_t) = \sigma^2(C_s, C_t), \tag{II.20}$$

É também sabido que [4]

$$\sigma\left(\frac{C_s}{C_t}\right) \cong \left(\frac{E[C_s]}{E[C_t]}\right)^2 \left(\frac{\sigma^2(C_s)}{E[C_s]^2} + \frac{\sigma^2(C_t)}{E[C_t]^2} - \frac{2\,\sigma^2(C_s, C_t)}{E[C_s]\,E[C_t]}\right), \tag{II.21}$$

substituindo as equações

$$\sigma\left(\frac{C_s}{C_t}\right) \cong \left(\frac{E[C_s]}{E[C_t]}\right)^2 \left(\frac{\sigma^2(C_s)}{E[C_s]^2} + \frac{\sigma^2(C_t)}{E[C_t]^2} - \frac{2\,\rho(C_s, C_t)\sigma(C_s)\sigma(C_t)}{E[C_s]\,E[C_t]}\right), \tag{II.22}$$

ou seja, minimizar $\sigma(C_s/C_t)$ é o mesmo que maximizar o coeficiente de correlação $\rho(C_s, C_t)$, conforme demonstrado em [13].

Como $C_s = C_{s1} + C_{s2} + \cdots + C_{sp}$,

$$\sigma^2(C_s) = \sum_{i=1}^{p} \sigma^2(C_{si}) + 2\sum_{i=1}^{p-1}\sum_{j=i+1}^{p} \sigma^2(C_{si}, C_{sj}), \tag{II.23}$$

Substituindo as covariâncias $\sigma^2(C_{si}, C_{sj})$ e as variâncias $\sigma^2(C_{si})$ em função das variâncias dos unitários $\sigma^2(C_{un})$,

$$\sigma^2(C_s) = p * \sigma^2(C_{un}) + 2\sigma^2(C_{un})\sum_{i=1}^{p-1}\sum_{j=i+1}^{p} \rho(C_{si}, C_{sj}), \tag{II.24}$$

analogamente,

$$\sigma^2(C_t) = q * \sigma^2(C_{un}) + 2\sigma^2(C_{un}) \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \rho(C_{ti}, C_{tj}), \tag{II.25}$$

e

$$\sigma^2(C_s, C_t) = \sum_{i=1}^{p} \sum_{j=1}^{q} \sigma^2(C_{si}, C_{tj}) = \sigma^2(C_{un}) \sum_{i=1}^{p} \sum_{j=1}^{q} \rho(C_{si}, C_{tj}), \tag{II.26}$$

De (II.18), (II.22), (II.23) e (II.24):

$$\rho(C_s, C_t) = \frac{\sum_{i=1}^{p} \sum_{j=1}^{q} \rho(C_{si}, C_{tj})}{\sqrt{\left(p + 2\sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \rho(C_{si}, C_{sj})\right)\left(q + 2\sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \rho(C_{ti}, C_{tj})\right)}} \tag{II.27}$$

Ainda segundo [4], o coeficiente de correlação entre dois componentes dentro de um circuito depende basicamente da distância entre eles e de um parâmetro relacionado ao processo de fabricação. Ou seja, conhecendo-se a variação aleatória sobre o valor nominal de um componente, é possível inferir mais sobre a variação assumida por um componente próximo a ele do que sobre o valor assumido por um componente distante.

Considerando uma matriz de $i$ linhas e $j$ colunas, o coeficiente de correlação entre os capacitores unitários $C_{i1,j1}$ e $C_{i2,j2}$ é dada por $\rho^D$, sendo $D$ a distância euclidiana entre os dois capacitores e $\rho$ uma constante dependente do parâmetro de processo. Para os capacitores $C_{i1,j1}$ e $C_{i2,j2}$, então, o coeficiente de correlação seria dado por $\rho^{\sqrt{(j_1 - j_2)^2 + (i_1 - i_2)^2}}$.

Portanto, outro objetivo a ser alcançado é maximizar o coeficiente de correlação entre os capacitores que necessitam estar bem casados. Neste trabalho, para um circuito que contenha $n$ capacitores será maximizada a média dos coeficientes de correlação entre eles

$$\bar{\rho}_C = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \rho_{Ci,Cj}, \tag{II.28}$$

de modo a priorizar igualmente todas as razões de capacitâncias.

# Capítulo 3

# Algoritmo de Posicionamento

## 3.1 – Função-Custo

O objetivo da ferramenta de CAD desenvolvida é organizar automaticamente os capacitores em uma matriz, minimizando os erros de descasamento entre eles. Para isso, é necessário um algoritmo de otimização. Quando utiliza-se tais algoritmos, o objetivo é maximizar ou minimizar uma determinada função, essa função é chamada de função-custo.

Por exemplo, no Problema do Caixeiro Viajante a função-custo a ser minimizada pelo algoritmo é o caminho percorrido pelo caixeiro em suas viagens e a variável do problema é a ordem das cidades a serem percorridas. No Problema da Mochila, por sua vez, a função-custo a ser maximizada seria a soma dos valores de todos os objetos carregados nela.

A função-custo utilizada na ferramenta deste projeto leva em consideração os coeficientes de correlação entre os capacitores que compõem um filtro a capacitores chaveados e o erro que o arranjo matricial desses capacitores apresenta em comparação com a geometria de centroide comum. Sua expressão matemática inicialmente proposta é

$$E = W_{centroid} * E_{centroid} + W_{corrCoef} * (1 - \bar{\rho}), \tag{III.1}$$

onde $W_{centroid}$, $W_{corrCoef}$, $E_{centroid}$ e $\bar{\rho}$ representam, respectivamente, o peso dado para o erro em relação a estrutura de centroide comum, o peso dado para a maximização da média dos coeficientes de correlação entre os capacitores, o erro em relação a estrutura de centroide comum e a média dos coeficientes de correlação entre os capacitores. É importante ressaltar que em (III.1) é utilizado $(1 - \bar{\rho})$ pois o quando a média dos coeficientes de correlação $\bar{\rho}$ é maximizada, $(1 - \bar{\rho})$ é minimizado. Além disso, como $0 < \bar{\rho} \leq 1$, $0 < 1 - \bar{\rho} \leq 1$ também, ou seja, esta é uma variável normalizada.

Para a realização deste cálculo, $W_{centroid}$ e $W_{corrCoef}$ são parâmetros do algoritmo, configurados pelo usuário, $E_{centroid}$ e $\bar{\rho}$ serão calculados, como é mostrado a seguir.

## 3.2 – Erro de Centroide Comum

O Erro de centroide comum $E_{centroid}$ deve medir, quantitativamente, o quão afastado um arranjo matricial de capacitores unitários está de um arranjo com centroide comum. Esse erro é dado pela soma dos erros de centroide de cada capacitor. A primeira etapa para o cálculo do erro é representar os capacitores da matriz através de coordenadas, posicionando a origem onde for mais conveniente, como mostra a Figura 3.1.

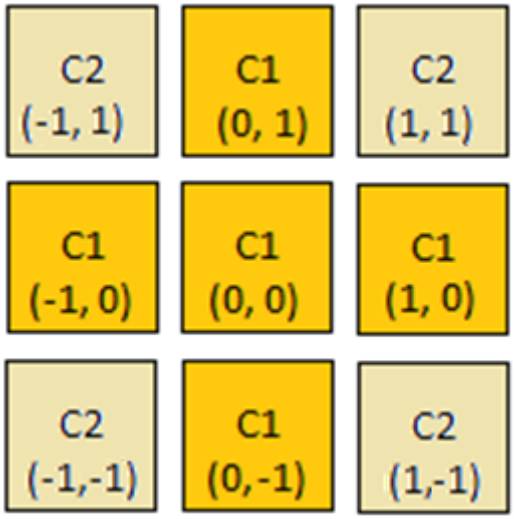


Figura 3.1 – Matriz de capacitores com as coordenadas.

A segunda etapa é somar as coordenadas de cada capacitor e dividir pelo número de capacitores unitários que o formam, de modo a obter as coordenadas de seu centroide. Para o capacitor C1 na Figura 3.1:

$$(x_1, y_1) = \frac{(0,1) + (-1,0) + (0,0) + (1,0) + (0,-1)}{5} = (0,0) \quad \text{(III.2)}$$

Para o capacitor C2:

$$(x_2, y_2) = \frac{(-1,1) + (1,1) + (-1,-1) + (1,-1)}{4} = (0,0) \quad \text{(III.3)}$$

No caso acima, o erro total de centroide seria nulo, pois, as coordenadas encontradas coincidem com o centro da matriz.

Porém, permutando os capacitores das coordenadas (0,-1) e (1,-1) de posição, é gerada a matriz mostrada na Figura 3.2.

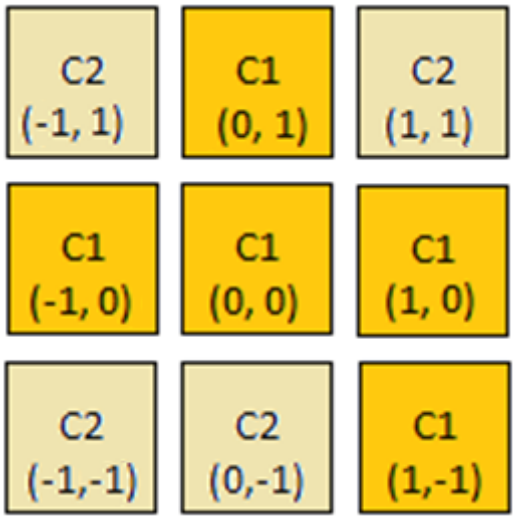


Figura 3.2 – Matriz de capacitores com as coordenadas.

Para o capacitor C1 dessa matriz:

$$(x_1, y_1) = \frac{(0,1) + (-1,0) + (0,0) + (1,0) + (1,-1)}{5} = \left(\frac{1}{5}, 0\right), \tag{III.4}$$

e, para o capacitor C2:

$$(x_2, y_2) = \frac{(-1,1) + (1,1) + (-1,-1) + (0,-1)}{4} = \left(\frac{-1}{4}, 0\right), \tag{III.5}$$

A equação do erro total de centroide é:

$$E_{centroid} = \frac{1}{Ncap * \eta} \sum_{i=1}^{Ncap} (x_i^2 + y_i^2), \tag{III.6}$$

onde, $Ncap$ representa o número de capacitores do circuito e $\eta$ é um fator de normalização para as dimensões da matriz, de modo que $0 \leq E_{centroid} \leq 1$ para qualquer matriz de capacitores unitários. Logo, para a configuração mostrada

$$E_{centroid} = \frac{1}{2\eta}(x_1^2 + y_1^2 + x_2^2 + y_{2)}^2. \tag{III.7}$$

Sendo $l$ o número de linhas e $c$ o número de colunas.

$$\eta = \left(\frac{l-1}{2}\right)^2 + \left(\frac{c-1}{2}\right)^2 = \frac{(l-1)^2 + (c-1)^2}{4}, \tag{III.8}$$

Para o exemplo mostrado:

$$\eta = \frac{(3-1)^2 + (3-1)^2}{4} = 2. \tag{III.9}$$

Fazendo os cálculos para a matriz acima: $E_{centroid} = 0{,}010625$.

Não há apenas uma matriz que torna nulo o erro de centroide. Dependendo do número de unitários e de suas dimensões, pode não haver nenhum arranjo possível em centroide comum. Porém, em alguns casos, pode ser que haja mais de um arranjo possível. O caso exemplo analisado acima é um deles. Para o número de unitários do exemplo, a matriz da Figura 3.3 também poderia ser escolhida:

| | | |
|---|---|---|
| C1 (-1, 1) | C2 (0, 1) | C2 (1, 1) |
| C1 (-1, 0) | C1 (0, 0) | C1 (1, 0) |
| C2 (-1,-1) | C2 (0,-1) | C1 (1,-1) |

Figura 3.3 – Matriz que apresenta erro de centroide comum nulo.

A solução da Figura 3.3 é menos trivial de ser encontrada, mas também apresenta erro de centroide comum nulo. Para estes casos, seria possível otimizar uma outra grandeza. Uma alternativa seria maximizar o coeficiente de correlação entre os capacitores, já que estes necessitam estar bem casados.

## 3.3 – Cálculo de Correlação

O segundo objetivo do programa será minimizar as variâncias entre as razões de componentes, ou seja, para um circuito contendo dois capacitores $C_1$ e $C_2$, ambos

implementados por associações em paralelo de unitários, a variância da razão $\sigma^2(C_2/C_1)$ deve ser mínima. Já para um circuito em que há três capacitores, a média das variâncias $\sigma^2(C_1/C_2)$, $\sigma^2(C_3/C_2)$ e $\sigma^2(C_2/C_3)$ deve ser mínima. Porém, em vez das variâncias serão utilizados os coeficientes de correlação, por se tratarem de parâmetros normalizado no intervalo entre zero e a unidade.

Para cada configuração de matriz testada no programa, será calculada a média dos coeficientes de correlação $\bar{\rho} = (2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\rho_{Ci,Cj})/(n(n-1))$, de modo a tornar unitário o valor máximo do parâmetro calculado. A fim a simplificar o modelo, as variáveis aleatórias correspondentes aos valores de capacitância serão consideradas igualmente distribuídas, onde os coeficientes de correlação entre os componentes dependerá apenas das distâncias:

$$\rho_{Ci,Cj} = \rho^{D}, \tag{III.10}$$

sendo $D$ a distância entre os capacitores unitários e $\rho$ um parâmetro relacionado ao processo de fabricação. O objetivo do programa é, então, maximizar $\bar{\rho}_C$, ou seja, torná-lo o mais próximo possível da unidade.

Para ilustrar, será calculado $\bar{\rho}_C$ para a matriz da Figura 3.2. Como mostrado na Sessão 2.6, o coeficiente de correlação independe das variâncias dos capacitores unitários, já que ela escala pelo mesmo fator todas as variâncias e covariâncias calculadas. Logo, será considerado aqui que essa variância é unitária, de modo a facilitar os cálculos. Assim, sabendo que o coeficiente de correlação entre duas variáveis aleatórias é dado por $\rho_{X,Y} = {\sigma_{X,Y}}^2/(\sigma_{X,X} * \sigma_{Y,Y})$, onde ${\sigma_{X,Y}}^2$ representa a covariância entre duas variáveis aleatórias $X$,$Y$, a Tabela 3.1 mostra a distância entre todos os capacitores unitários que formam $C_1$, necessários para o cálculo de ${\sigma_{C1,C1}}^2$:

| Coordenadas | Coordenadas | Distância |
|---|---|---|
| (0,1) | (0,1) | 0 |
| (0,1) | (-1,0) | $\sqrt{2}$ |
| (0,1) | (0,0) | 1 |
| (0,1) | (1,0) | $\sqrt{2}$ |
| (0,1) | (1,-1) | $\sqrt{5}$ |
| (-1,0) | (-1,0) | 0 |
| (-1,0) | (0,0) | 1 |
| (-1,0) | (1,0) | 2 |

| | | |
|---|---|---|
| (-1,0) | (1,-1) | $\sqrt{5}$ |
| (0,0) | (0,0) | 0 |
| (0,0) | (1,0) | 1 |
| (0,0) | (1,-1) | $\sqrt{2}$ |
| (1,0) | (1,0) | 0 |
| (1,0) | (1,-1) | 1 |
| (1,-1) | (1,-1) | 0 |

Tabela 3.1 – Distâncias para o cálculo de ${\sigma_{C1,C1}}^2$

$$\sigma_{C1,C1}{}^2 = 5\rho^0 + 2[(\rho^{\sqrt{2}} + \rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}}) + \left(\rho^1 + \rho^2 + \rho^{\sqrt{5}}\right) + \left(\rho^1 + \rho^{\sqrt{2}}\right) + (\rho^1)] \quad \text{(III.11)}$$

A Tabela 3.2 mostra as distâncias entre os capacitores unitários que formam C2, necessárias para o cálculo de ${\sigma_{C2,C2}}^2$:

| Coordenadas | Coordenadas | Distância |
|---|---|---|
| (-1,1) | (-1,1) | 0 |
| (-1,1) | (1,1) | 2 |
| (-1,1) | (-1,-1) | 2 |
| (-1,1) | (0,-1) | $\sqrt{5}$ |
| (1,1) | (1,1) | 0 |
| (1,1) | (-1,-1) | $2\sqrt{2}$ |
| (1,1) | (0,-1) | $\sqrt{5}$ |
| (-1,-1) | (-1,-1) | 0 |
| (-1,-1) | (0,-1) | 1 |
| (0,-1) | (0,-1) | 0 |

Tabela 3.2 – Distâncias para o cálculo de ${\sigma_{C2,C2}}^2$

$$\sigma_{C2,C2}{}^2 = 4\rho^0 + 2[(\rho^2 + \rho^2 + \rho^{\sqrt{5}}) + \left(\rho^{2*\sqrt{2}} + \rho^{\sqrt{5}}\right) + (\rho^1)] \quad \text{(III.12)}$$

O próximo cálculo a ser feito é ${\sigma_{C1,C2}}^2$. A Tabela 3.3 mostra as distâncias entre todos os capacitores C1 e C2.

| Coordenadas dos Capacitores C1 | Coordenadas dos Capacitores C2 | Distância |
|---|---|---|
| (0,1) | (-1,1) | 1 |
| (0,1) | (1,1) | 1 |
| (0,1) | (-1,-1) | $\sqrt{5}$ |

| | | |
|---|---|---|
| (0,1) | (0,-1) | 2 |
| (-1,0) | (-1,1) | 1 |
| (-1,0) | (1,1) | $\sqrt{5}$ |
| (-1,0) | (-1,-1) | 1 |
| (-1,0) | (0,-1) | $\sqrt{2}$ |
| (0,0) | (-1,1) | $\sqrt{5}$ |
| (0,0) | (1,1) | $\sqrt{2}$ |
| (0,0) | (-1,-1) | $\sqrt{2}$ |
| (0,0) | (0,-1) | 1 |
| (1,0) | (-1,1) | $\sqrt{5}$ |
| (1,0) | (1,1) | 1 |
| (1,0) | (-1,-1) | $\sqrt{5}$ |
| (1,0) | (0,-1) | $\sqrt{2}$ |
| (1,-1) | (-1,1) | $2\sqrt{2}$ |
| (1,-1) | (1,1) | 2 |
| (1,-1) | (-1,-1) | 2 |
| (1,-1) | (0,-1) | 1 |

Tabela 3.3 – Distâncias para o cálculo de ${\sigma_{C1,C2}}^2$

$$\sigma_{C1,C2}{}^2 = (\rho^1 + \rho^1 + \rho^{\sqrt{5}} + \rho^2) + (\rho^1 + \rho^{\sqrt{5}} + \rho^1 + \rho^{\sqrt{2}}) + (\rho^{\sqrt{5}} + \rho^{\sqrt{2}} + \rho^{\sqrt{2}} + \rho^1) + (\rho^{\sqrt{5}} + \rho^1 + \rho^{\sqrt{5}} + \rho^{\sqrt{2}}) + (\rho^{2\sqrt{2}} + \rho^2 + \rho^2 + \rho^1) \quad \text{(III.13)}$$

Finalmente, $\bar{\rho}_C = {\sigma_{C1,C2}}^2/(\sigma_{C1,C1} * \sigma_{C2,C2})$. A Tabela 3.4 mostra os valores de $\bar{\rho}_C$ para alguns valores do parâmetro $\rho$.

| Parâmetro $\rho$ | $\bar{\rho}_C$ |
|---|---|
| 0.4 | 0.6080 |
| 0.6 | 0.7942 |
| 0.8 | 0.9180 |

Tabela 3.4 – Valores de $\bar{\rho}_C$ para alguns valores de $\rho$.

Percebe-se que a quantidade de cálculos a serem feitos é muito grande, uma vez que seria necessário realizá-los para cada configuração de matriz testada. De fato, a complexidade do cálculo é $\theta(n^4)$. Felizmente, pode-se diminuí-la para $\theta(n^2)$, aproveitando os cálculos da matriz anterior caso realize-se uma permutação de dois

capacitores unitários na matriz. Para isso, é necessário guardar a matriz de covariâncias, que seria

$$M_{Cov} = \begin{bmatrix} \sigma_{C1,C1} & \sigma_{C1,C2} \\ \sigma_{C1,C2} & \sigma_{C2,C2} \end{bmatrix}. \quad \text{(III.14)}$$

Primeiramente, serão analisados $\sigma_{C1,C1}$, $\sigma_{C1,C2}$ e $\sigma_{C2,C2}$ para a matriz da Figura 3.3. A Tabela 3.5 mostra os cálculos para essa matriz.

| Coordenadas | Coordenadas | Distância |
|---|---|---|
| (-1,1) | (-1,1) | 0 |
| (-1,1) | (-1,0) | 1 |
| (-1,1) | (0,0) | $\sqrt{2}$ |
| (-1,1) | (1,0) | $\sqrt{5}$ |
| (-1,1) | (1,-1) | $2\sqrt{2}$ |
| (-1,0) | (-1,0) | 0 |
| (-1,0) | (0,0) | 1 |
| (-1,0) | (1,0) | 2 |
| (-1,0) | (1,-1) | $\sqrt{5}$ |
| (0,0) | (0,0) | 0 |
| (0,0) | (1,0) | 1 |
| (0,0) | (1,-1) | $\sqrt{2}$ |
| (1,0) | (1,0) | 0 |
| (1,0) | (1,-1) | 1 |
| (1,-1) | (1,-1) | 0 |

Tabela 3.5 – Distâncias para o cálculo de ${\sigma_{C1,C1}}^2$.

$${\sigma_{C1,C1}}^2 = 5\rho^0 + 2[(\rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}} + \rho^{2\sqrt{2}}) + \left(\rho^1 + \rho^2 + \rho^{\sqrt{5}}\right) + \left(\rho^1 + \rho^{\sqrt{2}}\right) + (\rho^1)] \quad \text{(III.15)}$$

A Tabela 3.6 mostra as distâncias entre os capacitores unitários que formam C2, necessárias para o cálculo de ${\sigma_{C2,C2}}^2$:

| Coordenadas | Coordenadas | Distância |
|---|---|---|
| (0,1) | (0,1) | 0 |
| (0,1) | (1,1) | 1 |
| (0,1) | (-1,-1) | $\sqrt{5}$ |
| (0,1) | (0,-1) | 2 |

| | | |
|---|---|---|
| (1,1) | (1,1) | 0 |
| (1,1) | (-1,-1) | $2\sqrt{2}$ |
| (1,1) | (0,-1) | $\sqrt{5}$ |
| (-1,-1) | (-1,-1) | 0 |
| (-1,-1) | (0,-1) | 1 |
| (0,-1) | (0,-1) | 0 |

Tabela 3.6 – Distâncias para o cálculo de ${\sigma_{C2,C2}}^2$

$${\sigma_{C2,C2}}^2 = 4\rho^0 + 2[(\rho^1 + \rho^{\sqrt{5}} + \rho^2) + \left(\rho^{2\sqrt{2}} + \rho^{\sqrt{5}}\right) + (\rho^1)]. \quad \text{(III.16)}$$

Enfim, ${\sigma_{C1,C2}}^2$. A Tabela 3.7 mostra as distâncias entre todos os capacitores C1 e C2.

| Coordenadas dos Capacitores C1 | Coordenadas dos Capacitores C2 | Distância |
|---|---|---|
| (-1,1) | (0,1) | 1 |
| (-1,1) | (1,1) | 2 |
| (-1,1) | (-1,-1) | 2 |
| (-1,1) | (0,-1) | $\sqrt{5}$ |
| (-1,0) | (0,1) | $\sqrt{2}$ |
| (-1,0) | (1,1) | $\sqrt{5}$ |
| (-1,0) | (-1,-1) | 1 |
| (-1,0) | (0,-1) | $\sqrt{2}$ |
| (0,0) | (0,1) | 1 |
| (0,0) | (1,1) | $\sqrt{2}$ |
| (0,0) | (-1,-1) | $\sqrt{2}$ |
| (0,0) | (0,-1) | 1 |
| (1,0) | (0,1) | $\sqrt{2}$ |
| (1,0) | (1,1) | 1 |
| (1,0) | (-1,-1) | $\sqrt{5}$ |
| (1,0) | (0,-1) | $\sqrt{2}$ |
| (1,-1) | (0,1) | $\sqrt{5}$ |
| (1,-1) | (1,1) | 2 |

| | | |
|---|---|---|
| (1,-1) | (-1,-1) | 2 |
| (1,-1) | (0,-1) | 1 |

Tabela 3.7 – Distâncias para o cálculo de ${\sigma_{C1,C2}}^2$

$$\sigma_{C1,C2}{}^2 = (\rho^1 + \rho^2 + \rho^2 + \rho^{\sqrt{5}}) + (\rho^{\sqrt{2}} + \rho^{\sqrt{5}} + \rho^1 + \rho^{\sqrt{2}}) + (\rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{2}} + \rho^1) + (\rho^{\sqrt{2}} + \rho^1 + \rho^{\sqrt{5}} + \rho^{\sqrt{2}}) + (\rho^{\sqrt{5}} + \rho^2 + \rho^2 + \rho^1) \quad \text{(III.17)}$$

A matriz da Figura 3.3 pode ser gerada a partir da matriz da Figura 3.2 permutando os capacitores das posições (-1,1) e (0,1), onde é possível perceber que somente os cálculos que envolvem as posições desses dois capacitores que se modifica de uma matriz para a outra. Sejam ${\sigma_{Cn,Cm}}^2{}_{3.2}$ e ${\sigma_{Cm,Cn}}^2{}_{3.3}$ as covariâncias entre os capacitores $Cm$ e $Cn$ das matrizes das Figuras 3.2 e 3.3, respectivamente.

$$\sigma_{C1,C1}{}^2{}_{3.3} - \sigma_{C1,C1}{}^2{}_{3.2} = \{5\rho^0 + 2[(\rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}} + \rho^{2\sqrt{2}}) + \left(\rho^1 + \rho^2 + \rho^{\sqrt{5}}\right) + \left(\rho^1 + \rho^{\sqrt{2}}\right) + (\rho^1)]\} - \{5\rho^0 + 2[(\rho^{\sqrt{2}} + \rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}}) + \left(\rho^1 + \rho^2 + \rho^{\sqrt{5}}\right) + \left(\rho^1 + \rho^{\sqrt{2}}\right) + (\rho^1)]\} \leftrightarrow \sigma_{C1,C1}{}^2{}_{3.3} - \sigma_{C1,C1}{}^2{}_{3.2} = 2 * [(\rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}} + \rho^{2\sqrt{2}}) - (\rho^{\sqrt{2}} + \rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}})] \quad \text{(III.18)}$$

$$\sigma_{C2,C2}{}^2{}_{3.3} - \sigma_{C2,C2}{}^2{}_{3.2} = \left\{4\rho^0 + 2[(\rho^1 + \rho^{\sqrt{5}} + \rho^2\right) + \left(\rho^{2\sqrt{2}} + \rho^{\sqrt{5}}\right) + (\rho^1)]\} - \{4\rho^0 + 2[(\rho^2 + \rho^2 + \rho^{\sqrt{5}}) + \left(\rho^{2\sqrt{2}} + \rho^{\sqrt{5}}\right) + (\rho^1)]\} \leftrightarrow \sigma_{C2,C2}{}^2{}_{3.3} - \sigma_{C2,C2}{}^2{}_{3.2} = 2[(\rho^1 + \rho^{\sqrt{5}} + \rho^2) - (\rho^2 + \rho^2 + \rho^{\sqrt{5}})] \quad \text{(III.19)}$$

Pelos cálculos acima, é possível perceber o padrão para atualizar os valores de variância.

C1 foi retirado da posição (0,1) e colocado na posição (-1,1). Então, para o cálculo de ${\sigma_{C1,C1}}^2{}_{3.3}$, é necessário calcular as distâncias entre o capacitor C1 na posição (0,1) e subtrair os coeficientes de correlação dele com os outros capacitores C1. Depois é necessário calcular as distâncias considerando C1 na posição (-1,1) e somar os coeficientes de correlação dele com os outros capacitores C1. Após isso, o valor encontrado é multiplicado por dois e somado na posição (1,1) de $M_{Cov}$. O procedimento é análogo para calcular ${\sigma_{C2,C2}}^2{}_{3.3}$, obtido o valor${\sigma_{C2,C2}}^2{}_{3.3} - \sigma_{C2,C2}{}^2{}_{3.2}$, atualiza-se a posição (2,2) da matriz.

A atualização da covariância ${\sigma_{C1,C2}}^2_{3.3}$ também é análoga, calculando-se ${\sigma_{C1,C2}}^2_{3.3} - {\sigma_{C1,C2}}^2_{3.2}$, percebe-se que apenas os cálculos que envolvem os capacitores das posições trocadas precisam ser refeitos, como mostrado em (III.21),

$$\begin{aligned}{\sigma_{C1,C2}}^2_{3.3} - {\sigma_{C1,C2}}^2_{3.2} = &\left\{\left(\rho^1 + \rho^1 + \rho^{\sqrt{5}} + \rho^2\right) + \rho^1 + \rho^{\sqrt{5}} + \rho^{\sqrt{5}} + \rho^{2\sqrt{2}}\right\} \\ &- \left\{\left(\rho^1 + \rho^2 + \rho^2 + \rho^{\sqrt{5}}\right) + \rho^{\sqrt{2}} + \rho^1 + \rho^{\sqrt{2}} + \rho^{\sqrt{5}}\right\},\end{aligned} \tag{III.21}$$

Calculam-se todas as distâncias entre os capacitores C2 e a posição (0,1), com as distâncias, são calculados os coeficientes de correlação e eles são subtraídos de ${\sigma_{C1,C2}}^2_{3.2}$. Após isso, são adicionados os coeficientes de correlação entre os capacitores C2 e o capacitor da posição (-1,1). Em seguida, são subtraídos os coeficientes de correlação entre os capacitores C1 e o capacitor da posição (-1,1) e, por fim, somados a ${\sigma_{C1,C2}}^2_{3.2}$ os coeficientes de correlação entre os capacitores C1 e o capacitor da posição (0,1).

O pseudocódigo para esse cálculo é descrito a seguir. (a1,b1) e (a2,b2) são as posições dos capacitores a serem trocadas, Mcov representa a matriz de covariância a ser atualizada, aqui a função tamanho calcula o tamanho de um vetor ou o número de linhas no caso de uma matriz. Caps representa a matriz de capacitores e ro representa o parâmetro para o cálculo de correlação, Ncap é o número de capacitores e Pbarra é o coeficiente de correlação médio.

```
INÍCIO
M = tamanho(Caps);
N = tamanho(Caps[1]);
PARA m = 1 até M FAÇA
    PARA n = 1 até N FAÇA
        p1 = p^(raiz((a1-m)^2+(b1-n)^2));
        p2 = p^(raiz((a2-m)^2+(b2-n)^2));

        SE ((m != a1) ou (n != b1)) ENTÃO

            SE (Caps[m][n] == Caps[a1][b1]) ENTÃO
                Mcov[Caps[m][n]][Caps[a1][b1]] += 2*(p2 - p1);

            SENÃO
                SE (!((m == a2) e (n == b2))) ENTÃO
                    Mcov[Caps[m][n]][Caps[a1][b1]] += p2 - p1;
                FIM SE
            FIM SE
            Mcov[Caps[a1][b1]][Caps[m][n]] =
Mcov[Caps[m][n]][Caps[a1][b1]];

        FIM SE
```

```
        SE ((m != a2) ou (n != b2)) ENTÃO

            SE (Caps(m,n) == Caps(a2,b2)) ENTÃO
                Mcov[Caps[m][n]][Caps[a2][b2]] += 2*(p2 - p1);

            SENÃO
                SE (!((m == a1) e (n == b1))) ENTÃO
                Mcov[Caps[m][n]][Caps[a2][b2]] += 2*(p2 - p1);
                FIM SE

            FIM SE
            Mcov[Caps[a1][b1]][Caps[m][n]] =
      Mcov[Caps[m][n]][Caps[a1][b1]];

        FIM SE

    FIM PARA
FIM PARA


// Calculando a soma dos coeficientes de correlação:
Pbarra = 0;
PARA n = 1 até (Ncap-1) FAÇA
    PARA m = (n+1) até Ncap FAÇA
        Pbarra += Mcov[m][n]/raiz(Mcov[n][n]*Mcov[m][m]);
    FIM PARA
FIM PARA

// Calculando a média dos coeficientes:
Pbarra = 2*Pbarra/(Ncap(Ncap-1));
```

## 3.4 – Simulated Annealing

O nome do algoritmo utilizado vem do processo de *annealing* ou recozimento. Na metalurgia, esta técnica consiste em aquecer um material sólido até a proximidade de seu ponto de fusão e resfriá-lo lentamente, de modo que haja tempo de os átomos assumirem uma configuração de menor energia e gerar uma configuração cristalina.

Sabe-se que se um material esquenta de uma temperatura $T_o$ até $T_1$ e depois é resfriado novamente até $T_o$, a energia termodinâmica do material, em geral, irá mudar, pois a energia termodinâmica não é uma variável de estado e é dependente da taxa de resfriamento. Uma taxa menor de resfriamento leva a uma maior variação de energia e, consequentemente, a um maior decrescimento da energia termodinâmica.

O *Simulated Annealing* [6] ou recozimento simulado é um algoritmo probabilístico, análogo ao processo de *annealing* da metalurgia, voltado para otimização. Para isso, o sistema a ser otimizado é imaginado como um sistema físico

imaginário cuja energia é dada por uma função, chamada de função-custo. Uma temperatura inicial **T** é escolhida e, a cada iteração, é resfriada, ou seja, multiplicada por um fator 0 < **alpha < 1**. Uma nova solução nas vizinhanças da solução atual é proposta a partir de uma pequena perturbação na solução atual, caso a nova solução seja melhor, ela é adotada, caso contrário, ela ainda tem uma chance de ser aceita. A taxa de aceitação é dada pela distribuição de probabilidades de Boltzmann, que é dependente da temperatura. A probabilidade de Boltzmann é dada por

$$P = e^{-\frac{\Delta E}{kT}}, \tag{III.22}$$

onde $k$ representa a constante de Boltzmann, $\Delta E$ representa a variação de energia e $T$ a temperatura.

Por se tratar de um algoritmo probabilístico, as soluções finais para as quais o algoritmo irá convergir nem sempre serão as mesmas, pois, assim como acontece na natureza, as configurações não são variáveis de estado da temperatura. Não há garantias nem de que estas serão ótimas. Porém alterando-se os parâmetros do algoritmo, é possível alterar a sua taxa de convergência e a chance de se obter uma solução ótima em um número finito de iterações. Estes parâmetros deverão ser configurados de acordo com o problema que está sendo resolvido.

O algoritmo *Simulated Annealing* é uma versão modificada do algoritmo criado por Nicholas Metropolis, publicado em 1953 [12] e geralmente é utilizado para soluções de problemas de classe NP-completos (problemas cuja complexidade para a resolução é não-polinomial), como o descrito no presente trabalho. A Tabela 3.8 e o fluxograma da Figura 3.4 mostram a estrutura do algoritmo.

| | |
|---|---|
| 1 | Escolha da solução inicial **solucao0**. |
| 2 | Atribuição da solução inicial a solução atual.<br>**solucao ← solucao0** |
| 3 | Guardar a melhor solução até o momento.<br>**melhorSolucao ← solução;** |
| 4 | Cálculo da função-custo atual<br>**funcaoCustoAtual ← funcaoCusto(solucao)** |
| 5 | Guardar a função-custo do melhor estado até o momento.<br>**melhorfuncaoCusto ← funcaoCustoAtual.** |

| | |
|---|---|
| 6 | Se a temperatura atual **T** é menor que a temperatura final estabelecida **Tf** ou a função-custo atual **funcaoCustoAtual** é menor que a máxima permitida, encerre o algoritmo. |
| 7 | Resfriar a temperatura atual.<br>**T ← alpha*T** |
| 8 | Perturbar a solução atual aleatoriamente a fim de gerar uma nova.<br>**novaSolucao ← perturba(solucao)** |
| 9 | Sendo rand() um número aleatório entre 0 e 1.<br>Se **funcaoCusto(novaSolucao) < funcaoCustoAtual ou exp(funcaoCustoAtual /T – funcaoCusto(novaSolucao)/T) > rand()**, execute o passo 10, senão, pule para o passo 15 |
| 10 | Atualizar a solução atual.<br>**solucao ← novaSolucao** |
| 11 | Atualizar a função-custo atual.<br>**funcaoCustoAtual ← funcaoCusto(novaSolucao)** |
| 12 | Se **funcaoCustoAtual < melhorFuncaoCusto**, execute o passo 13, senão, pule para o passo 15 |
| 13 | Atualizar a melhor solução atual.<br>**melhorSolucao ← solução** |
| 14 | Atualizar a função-custo.<br>**melhorFuncaoCusto ← funcaoCustoAtual**<br>Volte para o passo 6. |

Tabela 3.8 – Estrutura clássica do algoritmo.

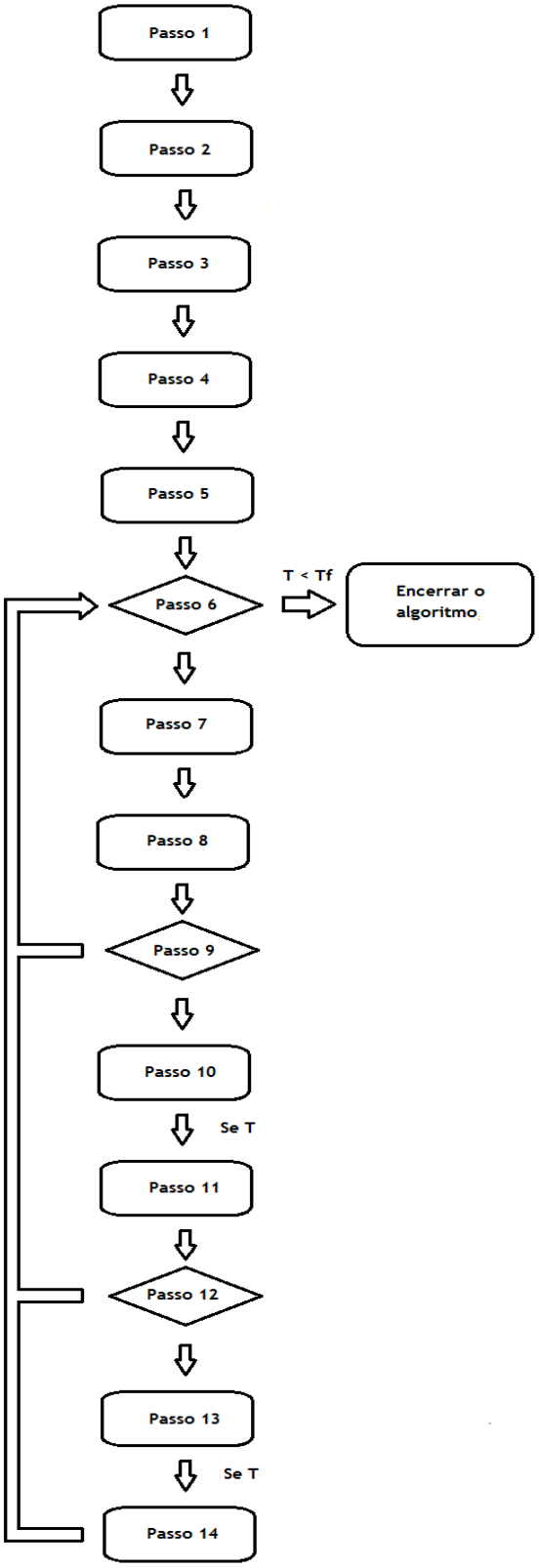


Figura 3.4 – Fluxograma do algoritmo

No passo 9 do algoritmo, pode parecer estranho ser possível aceitar soluções piores. Porém, isso é feito de modo a fugir de mínimos locais, caso o algoritmo aceitasse somente soluções melhores, o primeiro mínimo local encontrado seria a solução final.

Outro ponto importante a ser notado é que no início do algoritmo, a temperatura é maior, logo, há mais chance de que o número aleatório gerado seja menor que a exponencial de Boltzmann calculada, e consequentemente, há maior chance de soluções piores serem aceitas. Com a diminuição da temperatura, a chance de soluções piores serem aceitas também diminui (basta verificar que para uma variação unitária de energia, por exemplo, teria-se $\lim_{T\to 0} e^{-1/T} = 0$). Isso permite que o algoritmo explore melhor o espaço de busca nas primeiras iterações e venha a convergir para a solução de mínimo local – que pode ser, ou não, o mínimo global – nas últimas iterações, quando a temperatura é mais baixa.

## 3.4 – Constrained Simulated Annealing

Na equação (III.1) foi proposta uma função-custo em que são dados pesos às figuras de mérito a serem maximizadas ou minimizadas – o erro de centroide e a média dos coeficientes de correlação entre os capacitores. Porém, lembrando que o objetivo primário do programa é anular o erro de centroide, considere os dois casos dispostos na Tabela 3.9.

| | Caso 1 | Caso 2 |
|---|---|---|
| Peso para o erro em relação a estrutura de centroide | 0.7 | 0.7 |
| Erro em relação a estrutura de centroide comum | 0.1 | 0 |
| Peso para média dos coeficiente de correlação | 0.7 | 0.7 |
| Média dos coeficiente de correlação | 0.87 | 0.75 |
| Função-custo conforme definida em (III.1) | 0.161 | 0.175 |

Tabela 3.9 – Dois casos a serem considerados.

O caso 2, apesar de possuir erro de centroide comum nulo, apresenta uma função-custo com maior valor. Seria possível modificar os pesos dados, porém, aqui será adotado um procedimento diferente. Será aplicada uma penalidade à função-custo, caso o erro de centroide seja não-nulo, assim, a nova função-custo $E^*$ será definida como

$$E^* = W_{centroid} * \lambda * E_{centroid} + W_{corrCoef} * (1 - \bar{\rho}), \quad \text{(III.23)}$$

A versão do algoritmo em que aplicação de uma penalidade que eleva o valor da função-custo caso a restrição, que nesse caso é $E_{centroid} = 0$, não seja cumprida é chamada de *Constraint Simulated Annealing*. Contudo, seria difícil de determinar o valor de $\lambda$ manualmente. Valores grandes de $\lambda$ farão com que a diferença entre funções-custo de duas configurações matriciais vizinhas seja muito abruptas, dificultando a aceitação de uma solução pior, entretanto, valores pequenos de $\lambda$ não cumprem a função para a qual foram designados e matrizes com erro de centroide não-nulo poderão ser priorizadas em detrimento de matrizes com erro de centroide nulo. Por isso, $\lambda$ será variável. O valor inicial de $\lambda$ será dado por quem está executando o algoritmo, assim como o $\lambda_{fator}$, necessário para o seu cálculo, como será visto à frente, a partir disso, este parâmetro é automaticamente calculado pelo algoritmo. Deste modo, o algoritmo encontra o valor ótimo para o parâmetro $\lambda$ e o valor ótimo da função-custo considerando-o.

A cada temperatura analisada, será feito um número de permutações definido pelo usuário (chamado no programa desenvolvido de número de permutações por ciclo), como é necessário perturbar tanto a matriz como o parâmetro $\lambda$, este parâmetro será perturbado uma vez para cada temperatura analisada. Assim, supondo que o usuário definiu 40 permutações por ciclo, serão feitas 39 permutações na matriz e uma perturbação em $\lambda$ a cada ciclo de resfriamento.

A perturbação na matriz é feita trocando-se dois capacitores de lugar, ou seja, dada uma certa configuração, é gerada uma configuração vizinha; já a perturbação em $\lambda$ é feita calculando-se primeiramente

$$\emptyset = \lambda_{fator} * \frac{T}{T_0} * Erro, \quad \text{(III.24)}$$

da seguinte forma

$$\lambda \leftarrow \left| \lambda + \emptyset * \left( \frac{2 * rand()}{32767} - 1 \right) \right|, \quad \text{(III.25)}$$

$rand()$ é um número aleatório inteiro gerado entre 0 e 32767, portanto,$(2 * rand()/32767 - 1)$ estará, necessariamente, entre -1 e 1, dessa forma, o desvio máximo em $\lambda$ é $\emptyset$. A variável $Erro$ mede o desvio da estrutura em relação a configuração de centroide comum de uma forma ligeiramente diferente de $E_{centroid}$. Seu cálculo será explicado a seguir. Para isso, considere a matriz da Figura 3.5.

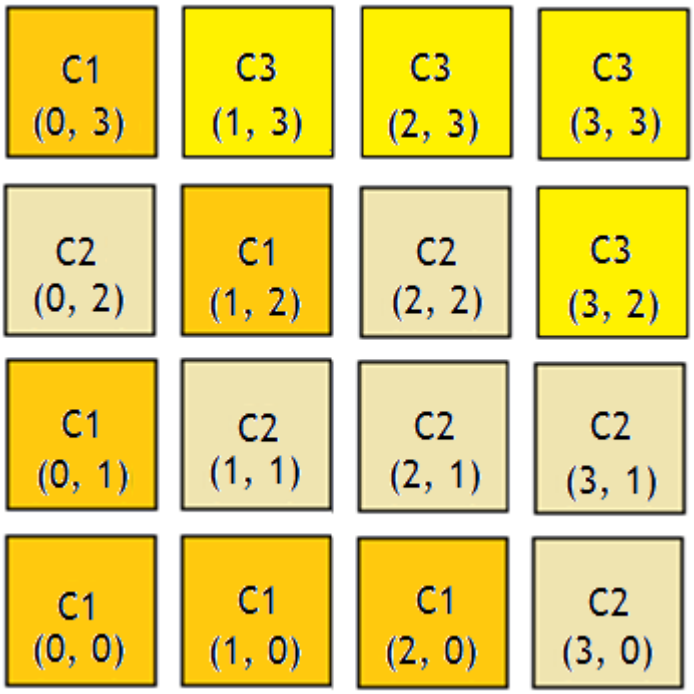


Figura 3.5 – Configuração Matricial para Análise da variável $Erro$.

Primeiramente, é suposto que cada capacitor tem valor unitário. Seja $Id$ o vetor com os valores dos capacitores divididos pelo valor de $C1$ $[C2\ C3\ C4\ ...\ Cn]/C1$ caso a variação de capacitância fosse nula nas duas direções, $Rx$ o mesmo vetor, porém, supondo uma variação de capacitância apenas na direção $x$ de 0.1 para cada coordenada e $Ry$ também $[C2\ C3\ C4\ ...\ Cn]/C1$, porém, supondo uma variação de capacitância apenas na direção $y$ de 0.1 para cada coordenada. Desse modo, primeiramente, serão somadas as coordenadas de cada capacitor,

$$(x_{C1}, y_{C1}) = (4, 6), \quad \text{(III.26)}$$

$$(x_{C2}, y_{C2}) = (11, 7), \quad \text{(III.27)}$$

$$(x_{C3}, y_{C3}) = (9, 11), \quad \text{(III.28)}$$

em seguida, $Id$ é calculado

$$Id = \frac{[6,4]}{6} \cong [1, 0.667], \tag{III.29}$$

$Rx$ e $Ry$ pode ser calculados com os vetores de somas das coordenadas dos capacitores e $Id$,

$$Rx = \frac{[6 + 0.1 * x_{C2}, 4 + 0.1 * x_{C3}]}{6 + 0.1 * x_{C1}} = \frac{[7.1,\ 4.9]}{6.4} = [1.109, 0.766], \tag{III.30}$$

$$Ry = \frac{[6 + 0.1 * y_{C2}, 4 + 0.1 * y_{C3}]}{6 + 0.1 * y_{C1}} = \frac{[6.7,\ 5.1]}{6.6} = [1.015, 0.773], \tag{III.31}$$

após esse procedimento, são calculados $Erro_x$ e $Erro_y$

$$Erro_x = \sum_{i=1}^{Ncap-1} \frac{(Id[i] - Rx[i])^2}{Id[i]^2} \cong \frac{(1.109 - 1)^2}{1.109^2} + \frac{(0.766 - 0.667)^2}{0.766^2} \cong 0.0264, \tag{III.32}$$

$$Erro_y = \sum_{i=1}^{Ncap-1} \frac{(Id[i] - Ry[i])^2}{Id[i]^2} \cong \frac{(1.015 - 1)^2}{1.015^2} + \frac{(0.773 - 0.667)^2}{0.773^2} \cong 0.0190, \tag{III.33}$$

$Erro$ é calculado como

$$Erro = \frac{Erro_x + Erro_y}{2 * (Ncap - 1)} \cong 0.0114. \tag{III.34}$$

# Capítulo 4

# Ferramenta Desenvolvida

## 4.1 – Plataforma de Desenvolvimento Utilizada

Foi necessário escolher a plataforma de desenvolvimento da ferramenta de CAD. Como o CAD desenvolvido é gráfico, um dos critérios de escolha foi a facilidade para a criação e manipulação de janelas.

Inicialmente, pensou-se em utilizar o *toolkit wxWidgets*, porém, como devido a algum conhecimento prévio em *Qt*, e ao fato de este *framework* possui uma documentação bastante extensa, escolheu-se utilizá-lo. Como a IDE (*Integrated Development Environment*) *Qt Creator* possui uma interface amigável, optou-se por utilizá-la no desenvolvimento do projeto.

## 4.2 – Classes Presentes no Software

Para o *Qt*, é utilizada a linguagem de programação C++, que é orientada a objetos, assim, o software foi desenvolvido utilizando classes e objetos. Há nove classes no programa, sendo quatro para implementação de janelas, uma para implementação do algoritmo, uma para implementação de uma barra de progresso e três classes auxiliares (uma para implementar a estrutura na qual é salva a configuração da matriz e seus parâmetros). Abaixo, elas são enumeradas e descritas:

- MainWindow

A classe *MainWindow* implementa a janela principal do programa, que pode ser vista na Figura 4.1.

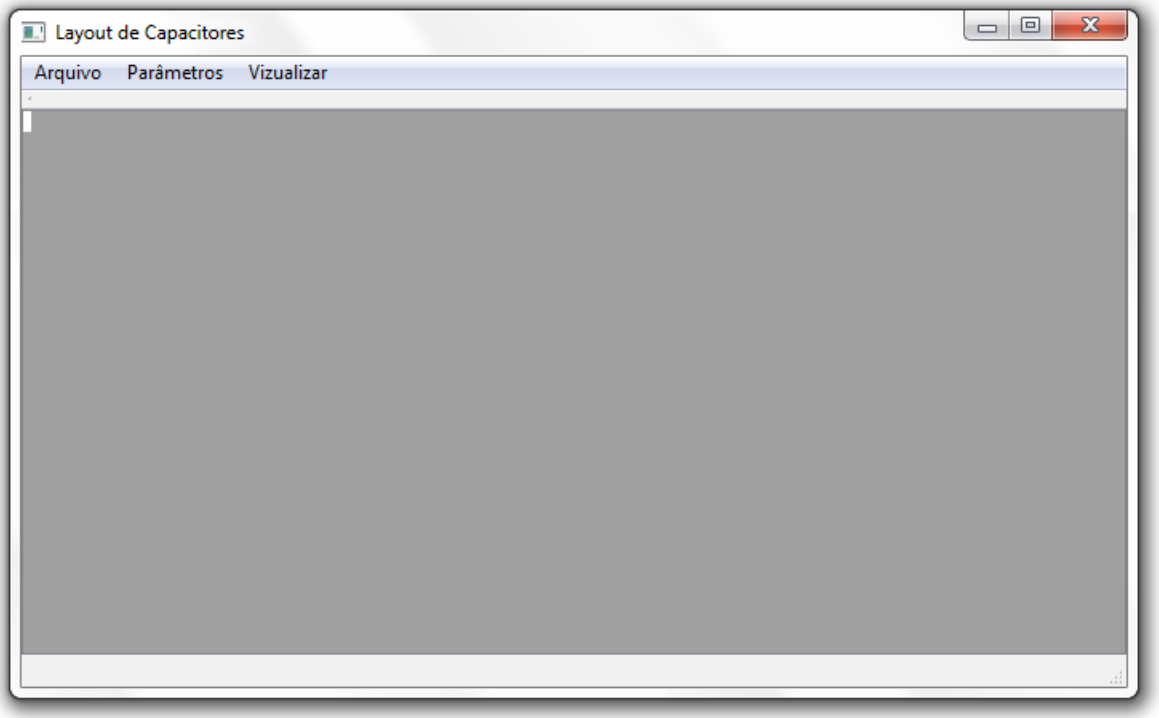


Figura 4.1 – Interface do programa

Ela é responsável por implementar todas as funções do menu principal, como abrir a janela de configuração de parâmetros e a janela de configuração da matriz. Além disso, quando os resultados do programa são gerados, eles são apresentados nessa janela. Estes resultados apresentados podem ser salvos em arquivo e os métodos para isso também são implementados nesta classe. Assim como o método para carregar parâmetros e matrizes através de arquivos.

- matrixConfigurationWindow

A classe *matrixConfigurationWindow*, como reflete o nome, é também uma janela, e sua função é configurar os dados da matriz. Esses dados são suas dimensões (número de linhas e número de colunas), o número de capacitores e o número de unitários que compõe cada capacitor. A classe auxiliar *InputValues* é utilizada nesta janela para mostrar o número de unitários que compõe cada capacitor, enquanto estes são configurados e a classe *auxValues* guarda os valores mostrados, que não são salvos enquanto o usuário não pressionar o botão Ok.

- InputValues

Quando o usuário do programa descreve os parâmetros da matriz de capacitores, é necessário informar o número de colunas, o número de linhas e o vetor de capacitores unitários, que contém a quantidade de unitários para cada capacitor da matriz. A classe *InputValues* é uma lista que guarda o capacitor que está sendo alterado na janela de configuração de matriz e os capacitores próximos a ele. Por exemplo, se a quantidade de unitários do capacitor $C_{11}$ está sendo alterada, o objeto da classe *InputValues* guardará o

número de unitários do $C_4$ ao $C_{18}$, cujos valores atuais serão mostrados na janela de configuração. A Figura 4.2 mostra um exemplo disso.

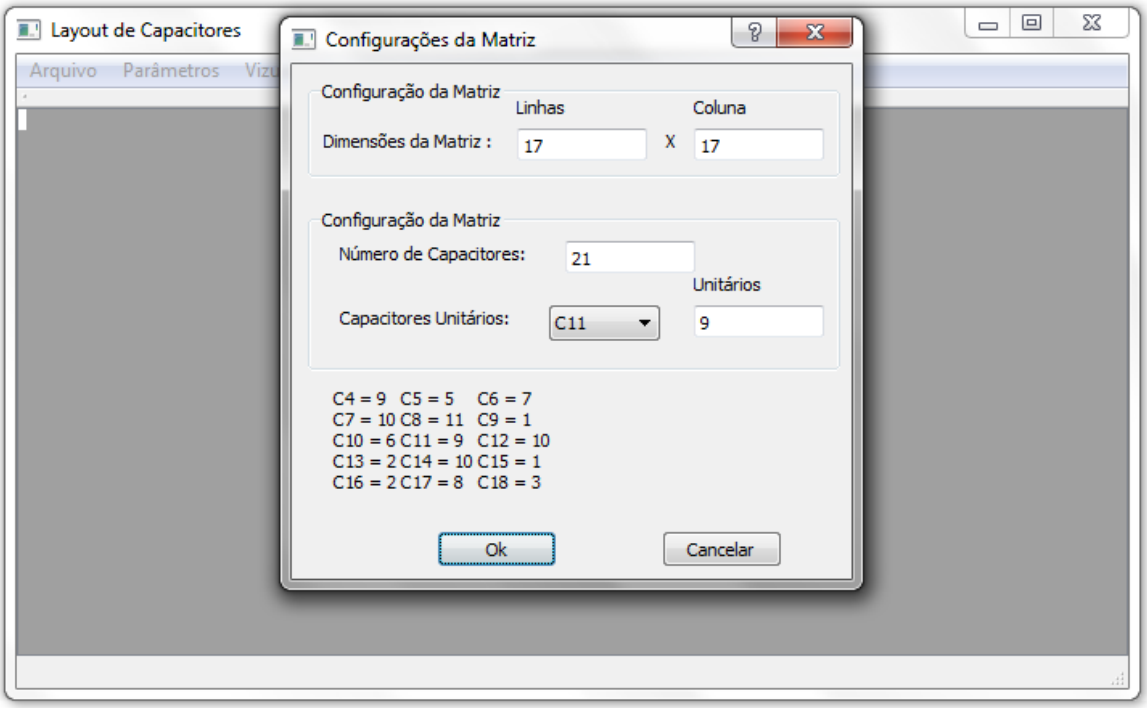


Fig 4.2 – Janela de Configuração de Matriz e utilização da classe InputValues

- auxValues

Classe auxiliar, guarda as configurações da matriz mostradas na janela de configuração de matriz, que só podem ser salvas quando o usuário pressiona o botão Ok.

- cap_matrix

A classe *cap_matrix* é utilizada no cálculo do algoritmo, todos os dados referentes às matrizes são guardados nesta classe (configuração atual, coeficiente de correlação atual, erro de centroide atual, custo total, melhor configuração encontrada até o momento, menor valor da função custo encontrado até o momento, maior coeficiente de correlação e menor erro de centroide, para nomear alguns). Alguns destes dados são utilizados nos métodos da própria classe e alguns são utilizados na classe *SimulatedAnnealing*.

- errorWindow

Classe que implementa a janela de erro a ser mostrada em caso de erro no programa. A Figura 4.3 ilustra uma situação em que isso acontece.

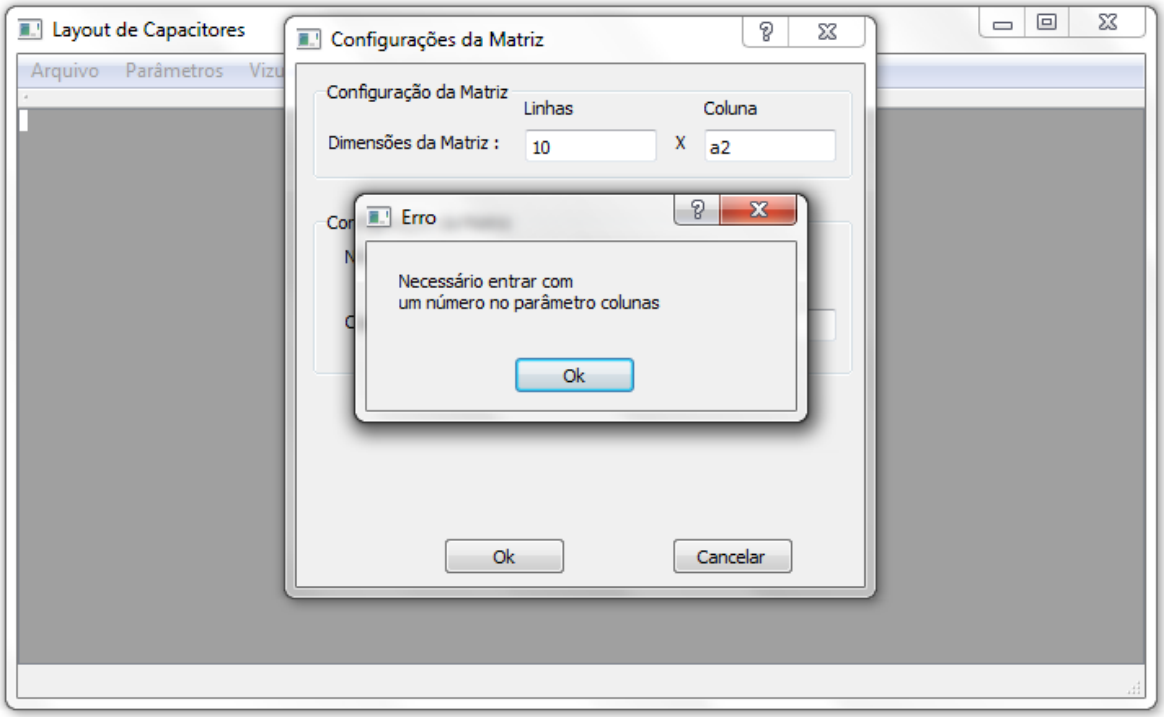


Figura 4.3 – Janela de erro mostrando que o parâmetro coluna não foi configurado corretamente

- parametersConfigurationWindow

A janela de configuração de parâmetros do algoritmo, implementada nesta classe, é mostrada na Figura 4.4.

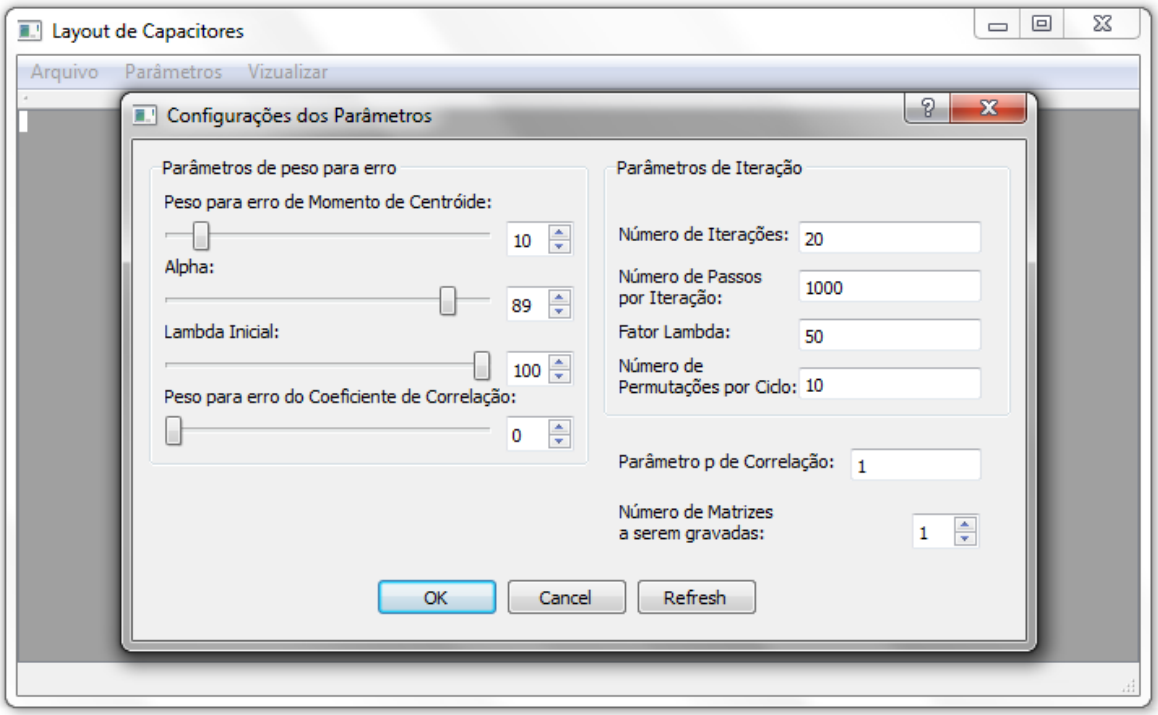


Figura 4.4 – Janela de Parâmetros

A Tabela 4.1 mostra a descrição de cada parâmetro a ser configurado. Os valores da Figura 4.4 são os valores padrão, valores pré-definidos, caso não se queira configurar.

| | |
|---|---|
| Número de iterações | Número de vezes que o algoritmo de otimização é executado. |
| Número de passos por iteração | Na Tabela 3.9, é definido que o algoritmo termina quando a temperatura alcança a temperatura final **Tf** desejada, aqui, alternativamente, será definido o número de ciclos do algoritmo. |
| Lambda inicial | Valor inicial de $\lambda$, utilizado na função-custo. Um Lambda inicial maior faz com que a penalidade dada ao Erro de centroide comum seja maior, no começo, como explicado no Capítulo 3, o valor inicial é dado pelo usuário do programa. |
| Número de permutações por ciclo | Número de permutações testadas para cada temperatura. |
| Peso para erro de Momento de Centróide | Quanto maior este parâmetro, mais o algoritmo tende a minimizar o erro de Momento de Centroide. O valor na janela é expresso em porcentagem. Este é o peso $W_{centroid}$ definido na função-custo do Capítulo 3. |
| Peso para erro do Coeficiente de Correlação | Quanto maior este parâmetro, mais o algoritmo tende a minimizar o erro de Coeficiente de Correlação, ou seja, tenta maximizar a média dos Coeficientes de Correlação. O valor na janela é expresso em porcentagem. Este é o peso $W_{corrCoef}$ definido na função-custo do Capítulo 3. |
| Alpha | Taxa de resfriamento. A cada ciclo do algoritmo, **T**←**alpha***T**. Quanto maior o |

| | |
|---|---|
| | alpha, mais rápido o resfriamento. |
| Fator Lambda | A função-custo de cada matriz é proporcional a $\lambda$, porém, este parâmetro não é mantido constante ao longo do algoritmo. Este representa o $\lambda_{fator}$ do Capítulo 3, responsável pela perturbação em $\lambda$. |
| Número de matrizes a serem gravadas | O programa grava a melhor matriz para cada execução, ou seja, se o número de iterações é configurado como 20, por exemplo, serão gravadas 20 matrizes, então, o usuário pode escolher mostrar até esse número de matrizes desenhadas após a simulação. Caso o usuário salve-as salve o resultado das simulações em arquivo, esse será o número de matrizes salvas. Tanto na tela quanto no arquivo, elas são mostradas por ordem crescente de custo. |
| Parâmetro p de correlação | No modelo de correlação espacial utilizado, o coeficiente de correlação é modelado como $\rho^D$, sendo $D$ a distância entre os capacitores, o parâmetro $\rho$ do qual a correlação depende é definido aqui, limitado no intervalo (0,1]. Quando $\rho = 1$, o coeficiente de correlação se torna constante e igual a 1, ou seja, independente da distância. Utilizando este valor, o cálculo do coeficiente de correlação não é levado em consideração. |

Tabela 4.1 – Parâmetros do algoritmo

- progressBar

A classe *progressBar* apenas cria uma barra de porcentagem e define seu tamanho, que é mostrada na janela enquanto as melhores soluções são calculadas. Ela é mostrada na Figura 4.5.

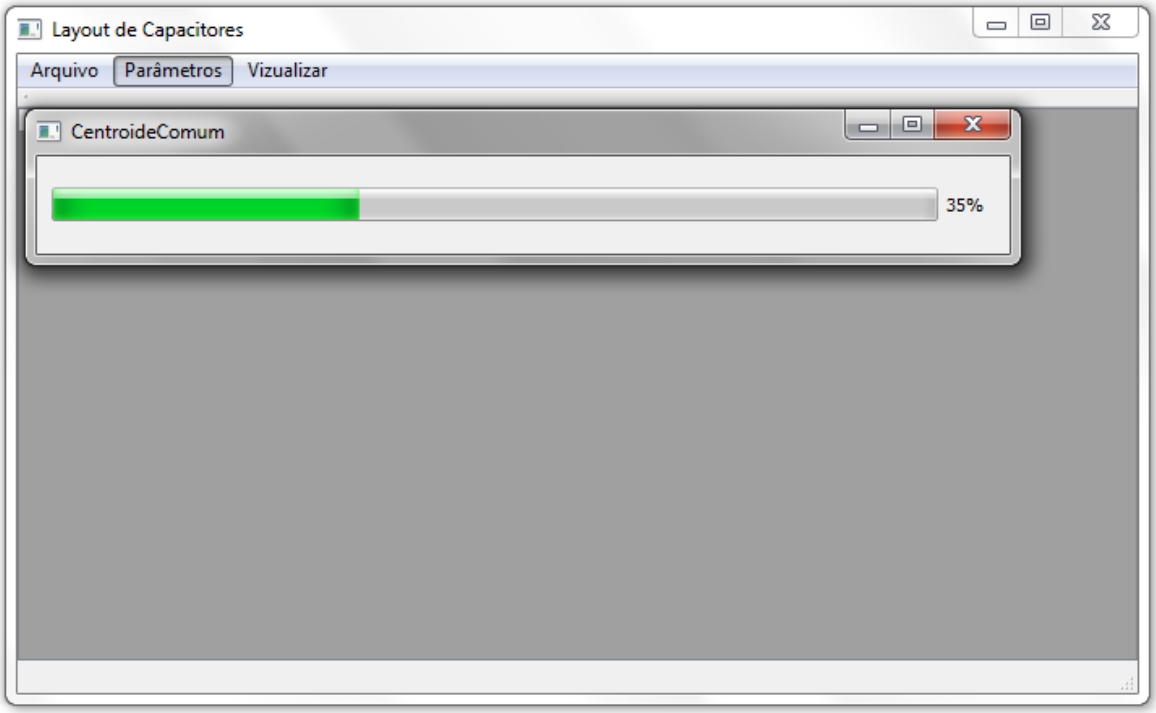


Figura 4.5 – Barra de Progresso.

- SimulatedAnnealing

*SimulatedAnnealing* é a classe na qual o algoritmo é realmente implementado, essa classe necessita dos parâmetros do algoritmo e da matriz. Os métodos implementados nesta classe *são temperatureSchedule*, para calcular a temperatura inicial do algoritmo; *algorithm*, para executar o algoritmo que calcula a melhor solução; *calculateLoadingBar*, que mostra o quanto do algoritmo já foi calculado e *statistic*, para guardar a média e a variância de custos, coeficientes de correlação e erro de centroide para as matrizes.

# Capítulo 5

# Resultados e Casos de Uso

## 5.1 – Resultados Obtidos

Abaixo são mostrados os resultados obtidos para algumas matrizes, para verificar o funcionamento do programa.

A Figura 5.1 mostra o resultado gerado pelo programa para uma matriz com quatro linhas e quatro colunas, contendo quatro capacitores: C1 – formado por quatro unitários em paralelo, C2 – formado por quatro unitários, C3 – formado por dois e C4 – formado por seis. Os parâmetros utilizados foram os padrões, considerando apenas o erro de centroide.

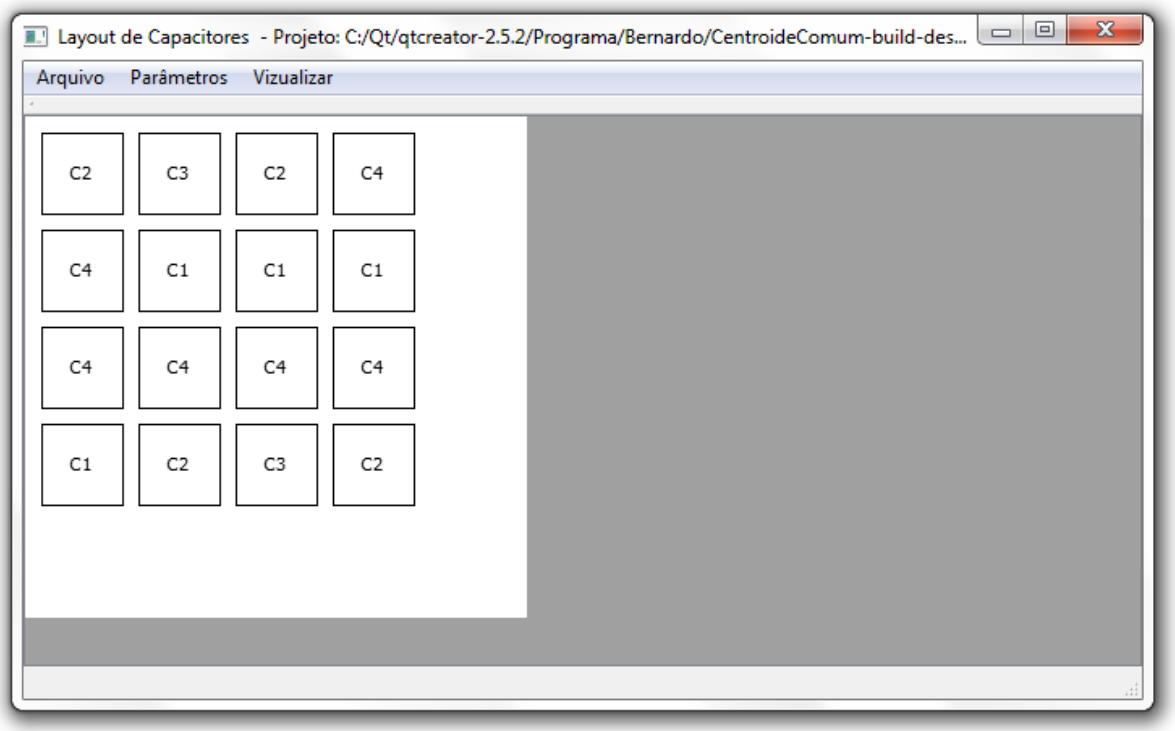


Figura 5.1 – Resultado gerado pelo programa

Salvando em arquivo a partir do menu no menu **Arquivo** → **Salvar**, é possível verificar que, de fato, a solução encontrada possui erro de centroide comum nulo. Como mostra a Figura 5.2.

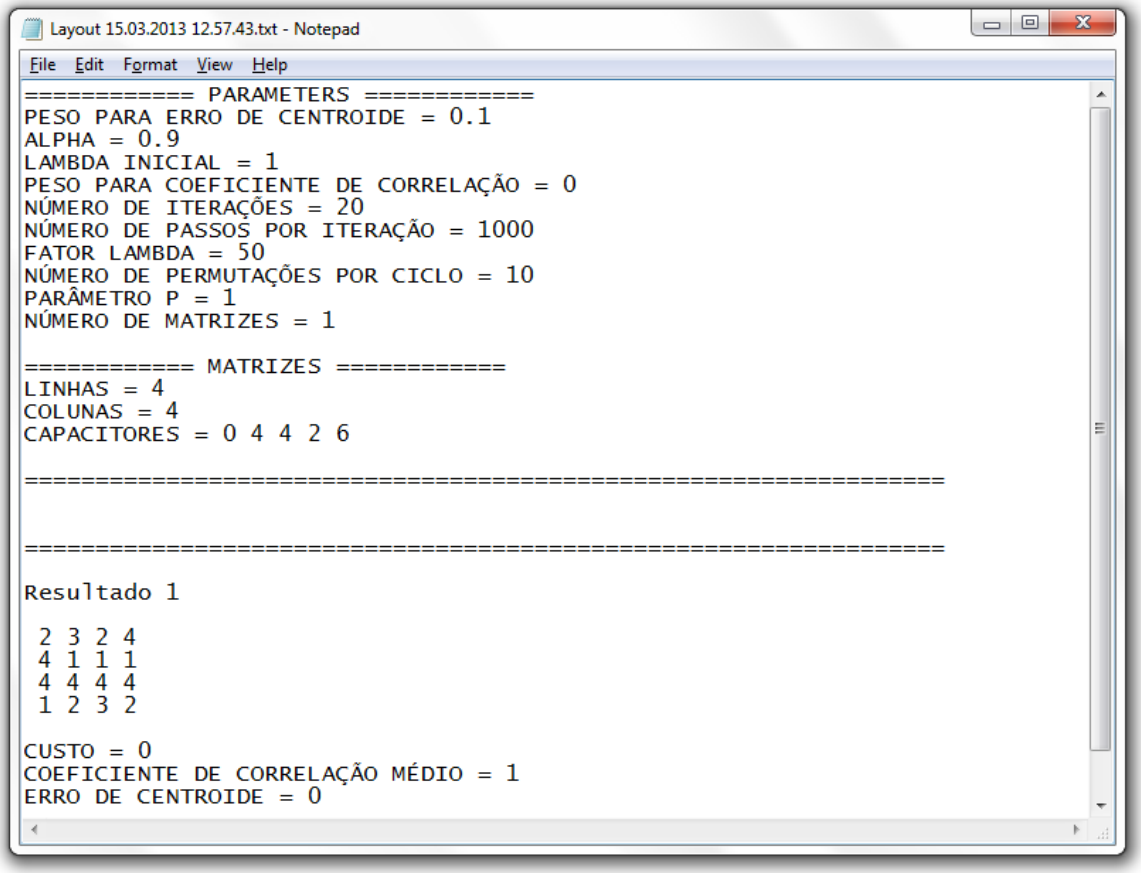


Figura 5.2 – Resultados salvos em arquivo.

O próximo exemplo ilustra uma matriz de dimensões oito por seis. Os capacitores nela presentes são: C1 e C2 – formados por quatro unitários cada, C3 – formado por cinco unitários, C4 – formado por sete unitários, C5 – formado por sete unitários e C6 – formado por treze unitários. O único parâmetro reconfigurado para este teste foi o número de número de iterações, do valor padrão 20 para 50. A ferramenta de zoom no menu **Visualizar** → **Zoom** para que a matriz pudesse ser visualizada inteiramente sem a utilização da barra de rolagem. A Figura 5.3 mostra a matriz gerada.

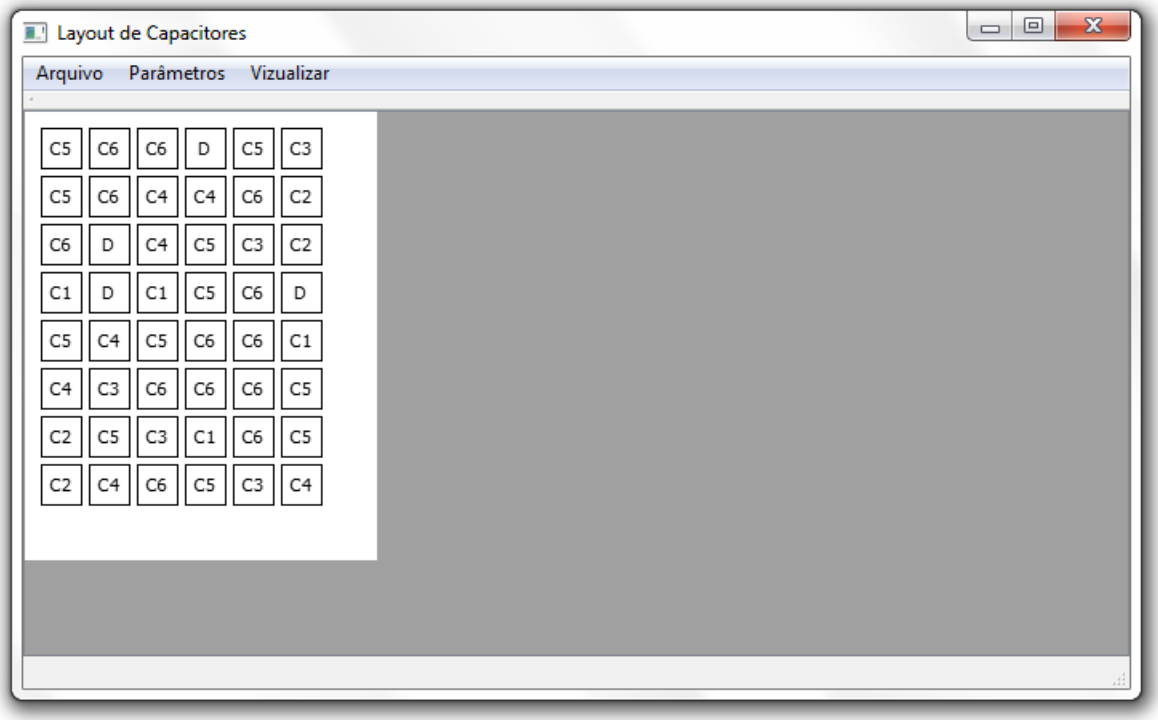


Figura 5.3 – Matriz gerada.

Este resultado apresenta erro de centroide $\cong 0{,}01521$. Nessa matriz, o número total de capacitores não é suficiente para preenche-la inteiramente, assim, é necessário o uso de *dummies*, no programa desenvolvido, eles são identificados na figura pela letra *D*.

Para a mesma matriz, configurando o número de passos por iteração como 3000, alpha como 0,95 e o número de iterações como 200, o melhor resultado da simulação apresenta erro de centroide $\cong 0{,}01267$. Menor que o anterior, mas apenas se fossem testadas todas as possibilidades, seria possível verificar se o erro encontrado é o mínimo para a matriz. Nesse caso, para a matriz com os capacitores unitários acima haveria duas soluções – tentar ajustar os parâmetros do algoritmo ou redimensionar a matriz. É importante notar que para o caso apresentado acima, seria bastante difícil obter uma solução com erro de centroide nulo, pois muitos dos capacitores são formados por um número primo de unitários. Redimensionando-a para sete linhas e sete colunas, uma com erro de centroide comum nulo foi encontrada. Como mostra a Fig 5.4.

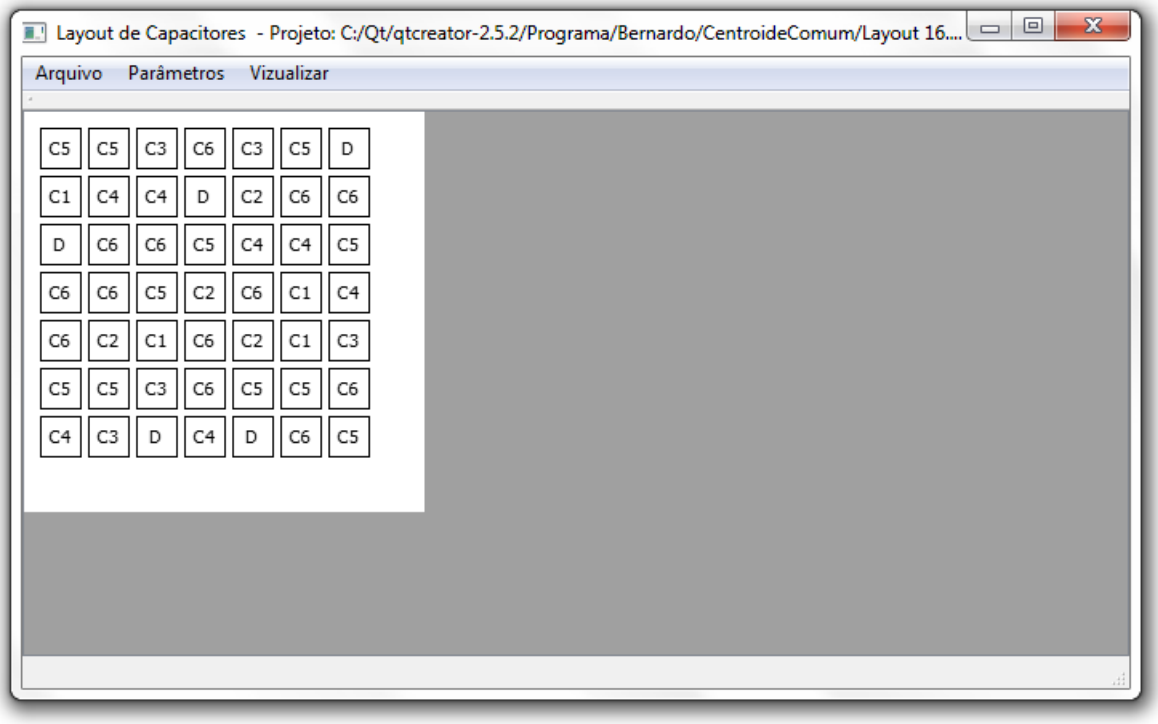


Figura 5.4 – Solução com erro de centroide nulo.

Nesse caso, foi considerado $\rho = 0.85$, a matriz acima apresenta uma média dos coeficientes de correlação = 0.9340, um valor alto considerando que o máximo valor é 1, e este só poderia ser alcançado caso $\rho$ também fosse unitário.

## 5.2 – Exemplo de Uso

Abaixo é mostrado como utilizar o programa.

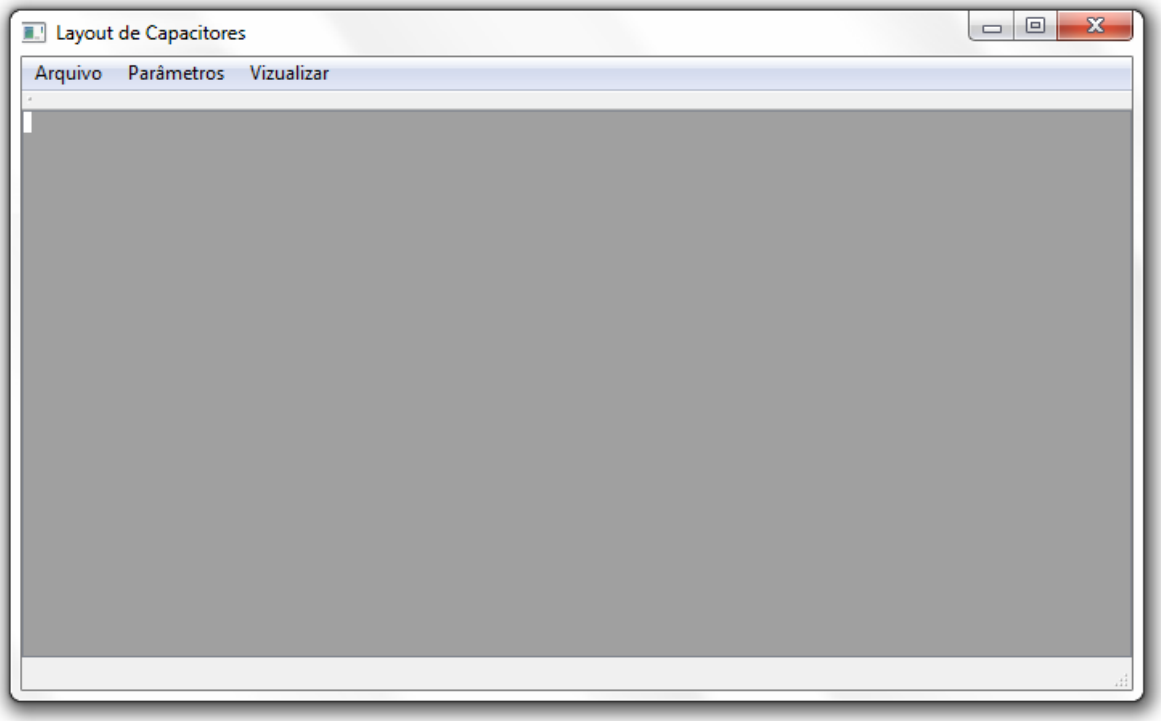


Figura 5.5 – Interface do Programa.

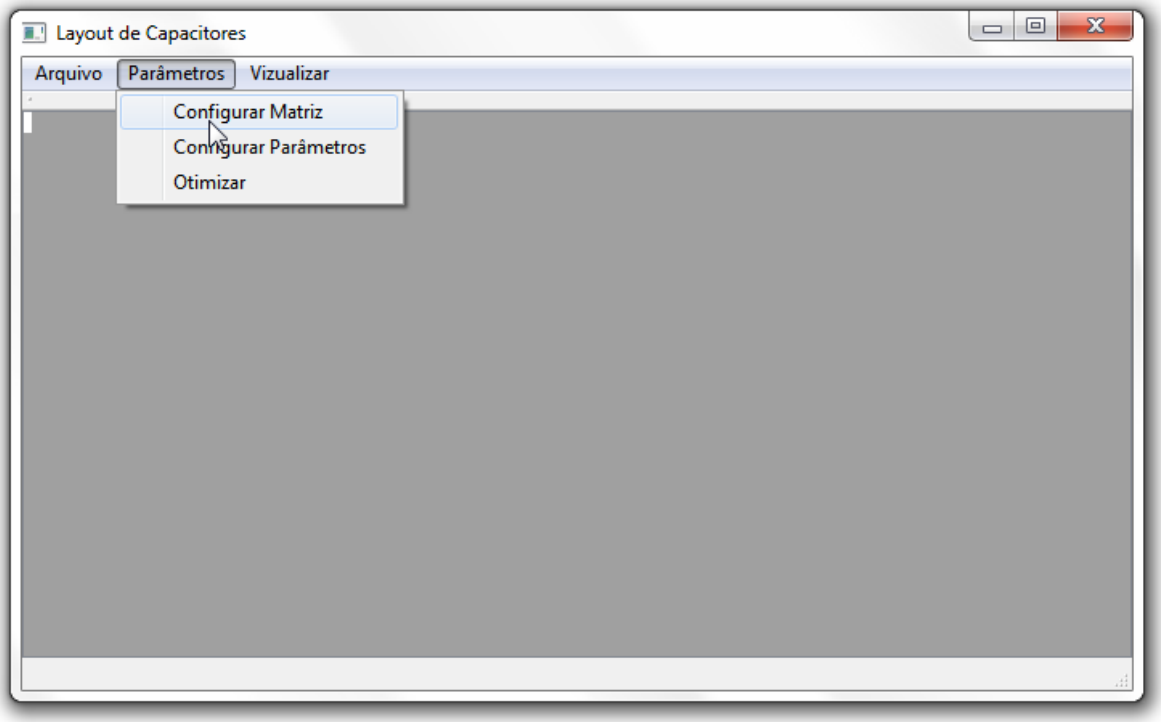


Figura 5.6 – Utilize a opção de menu **Parâmetros → Configurar Matriz** para adicionar os dados da matriz de capacitores a ser gerada.

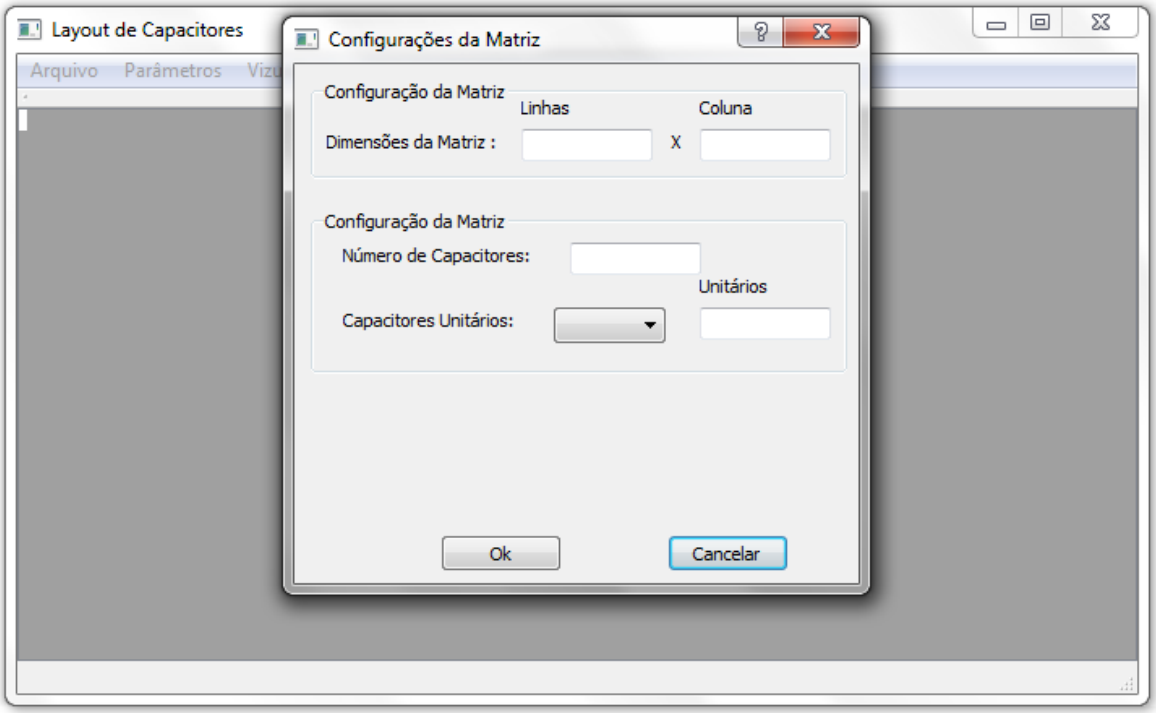


Figura 5.7 – Após clicar em **Configurar Matriz**, essa janela é aberta.

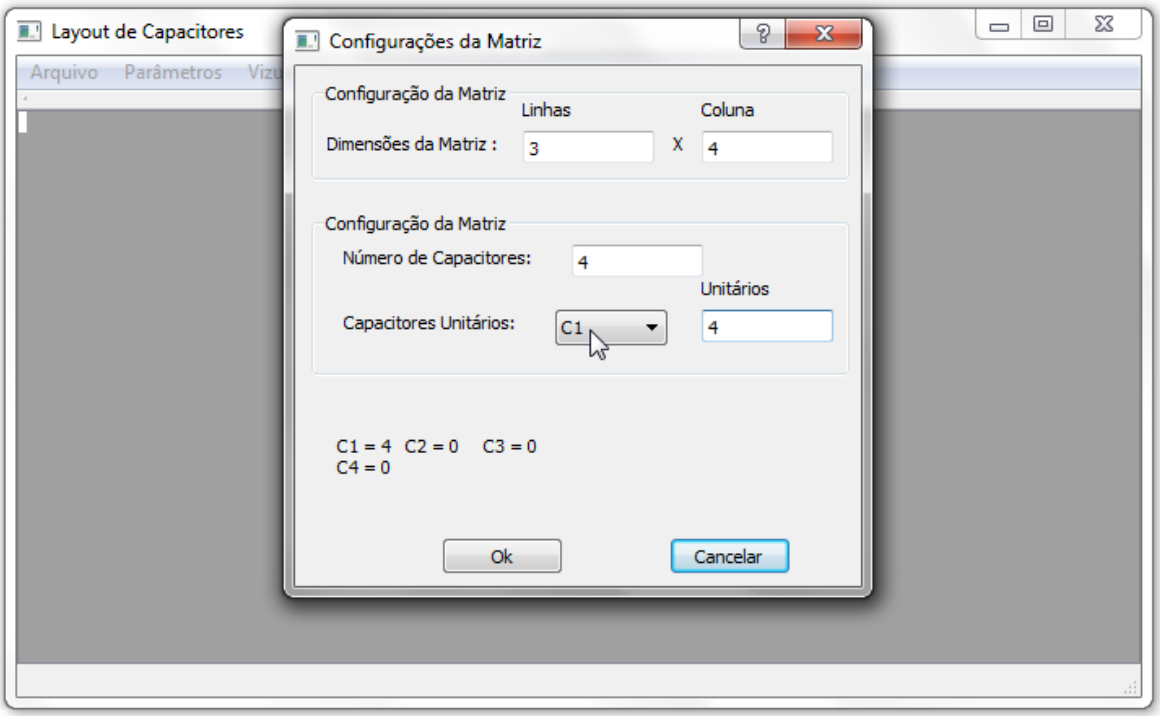


Figura 5.8 – Informe o número de Linhas e de Colunas da matriz de capacitores e Número de Capacitores. Após configurar o Número de Capacitores, a *combobox* onde aponta o cursor muda, contendo todos os capacitores que serão posicionados na matriz. O número de unitários em paralelo para realizar o capacitor C1, por exemplo, foi definido como 4.

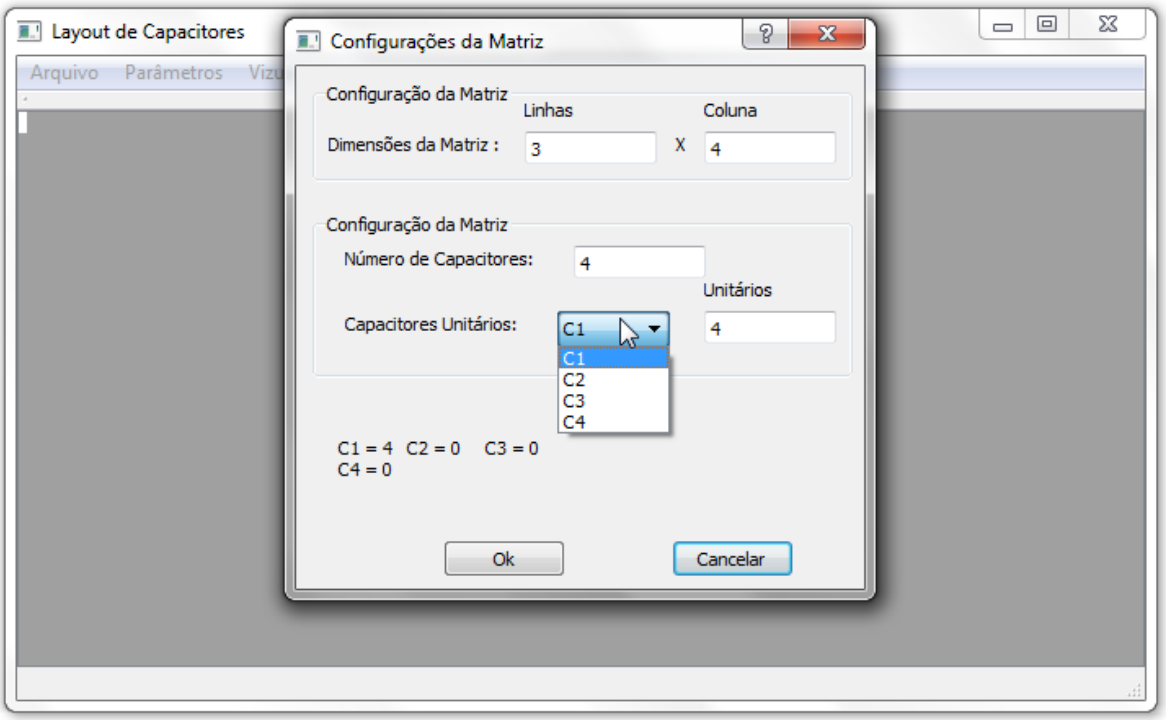


Figura 5.9 – *Combobox* expandida, 4 capacitores, conforme definido. O usuário deve selecionar o capacitor cujo número de unitários será configurado.

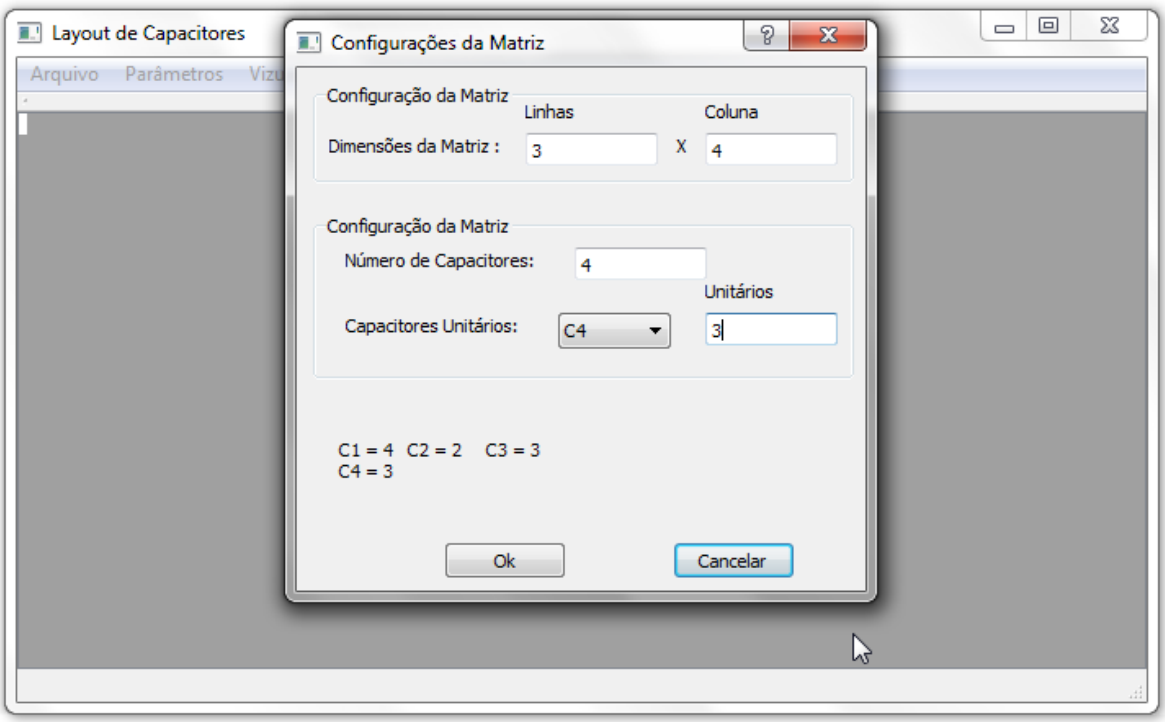


Figura 5.10 – Definido o número de unitários para cada um dos capacitores, uma lista com o número de unitários configurados para cada capacitor é apresentada na parte inferior da janela.

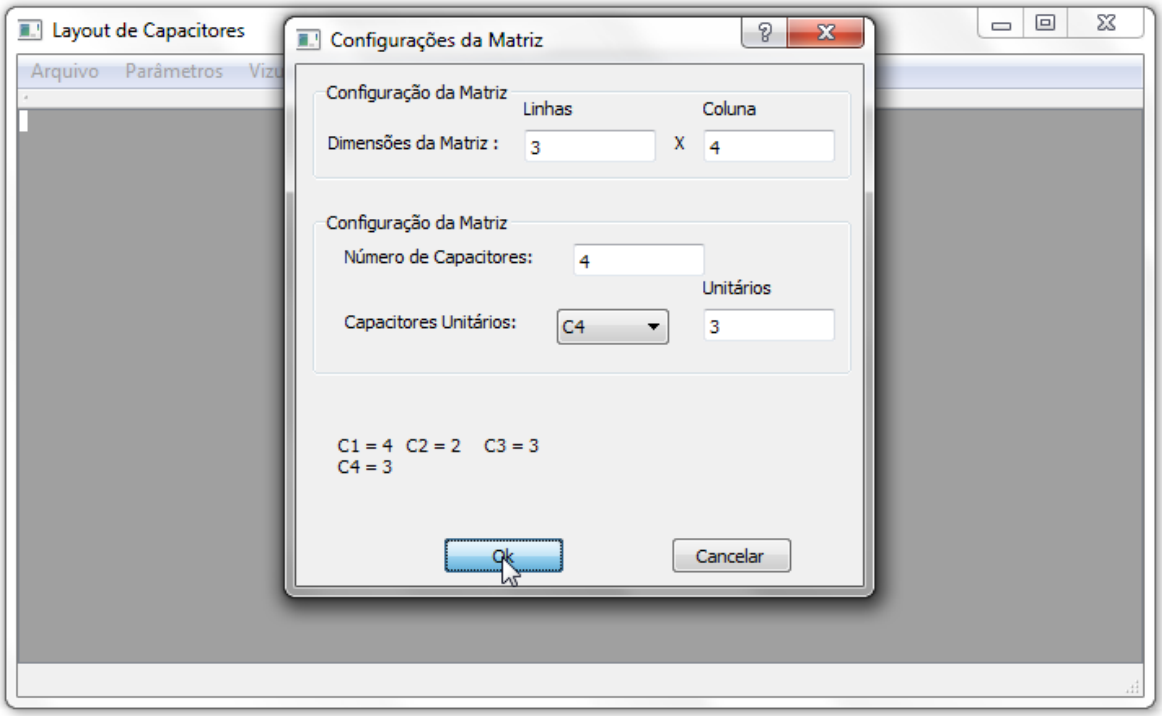


Figura 5.11 – Para finalizar, clica-se em Ok. Após isso, o programa retorna a tela inicial

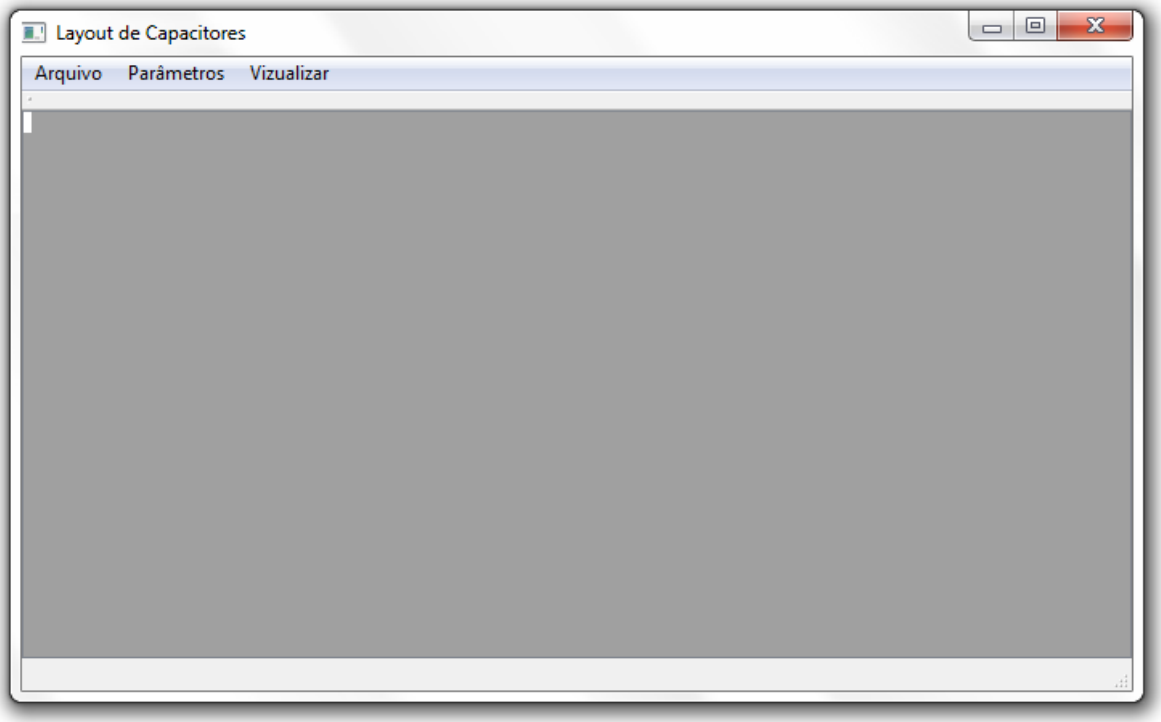


Figura 5.12 – Retorno a tela inicial

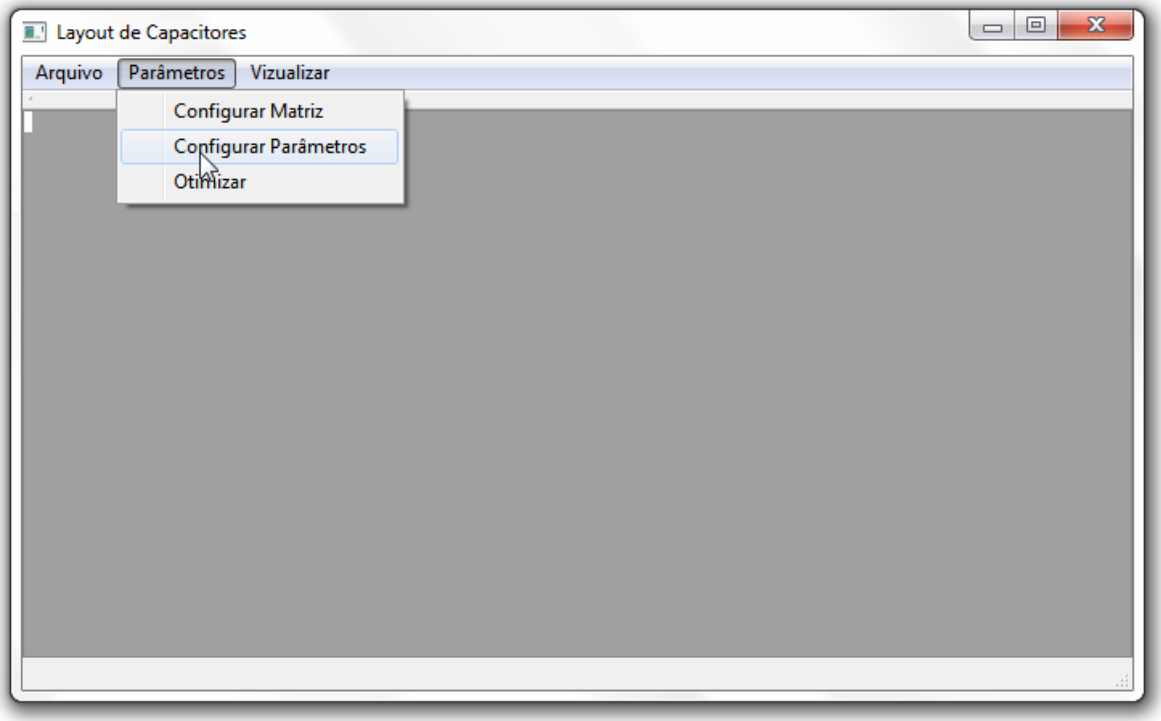


Figura 5.13 – Para configurar os parâmetros do algoritmo de otimização, o usuário deve selecionar o menu **Parâmetros→ Configurar Parâmetros**

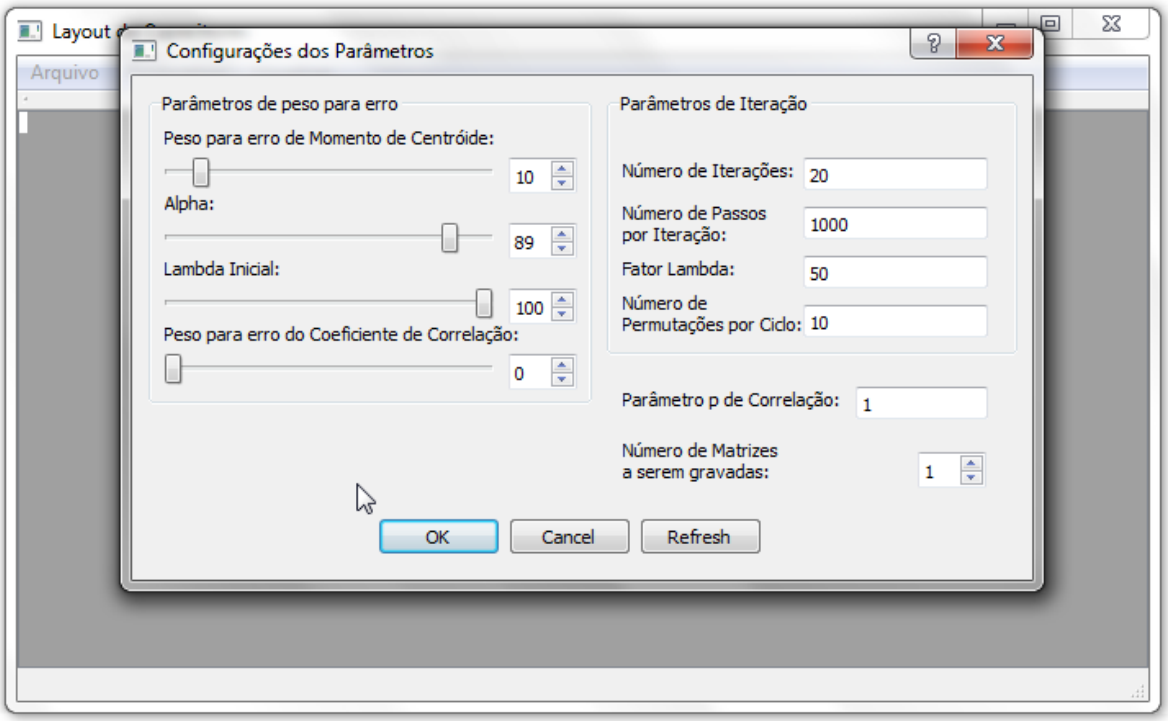


Figura 5.14 – Janela de Configuração de Parâmetros aberta, valores padrão mostrados.

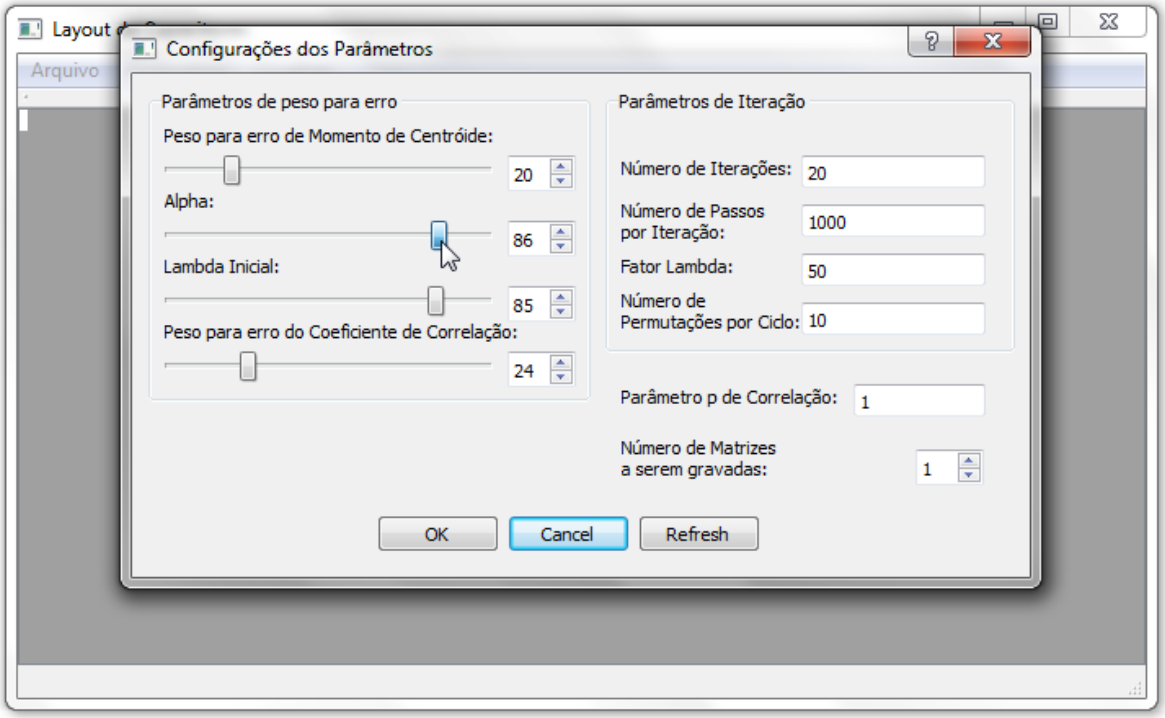


Figura 5.15 – Alteração de alguns parâmetros, para os parâmetros do lado esquerdo, é possível alterá-los por meio do *slider*, como mostrado ou digitando-se o número na caixa ao lado.

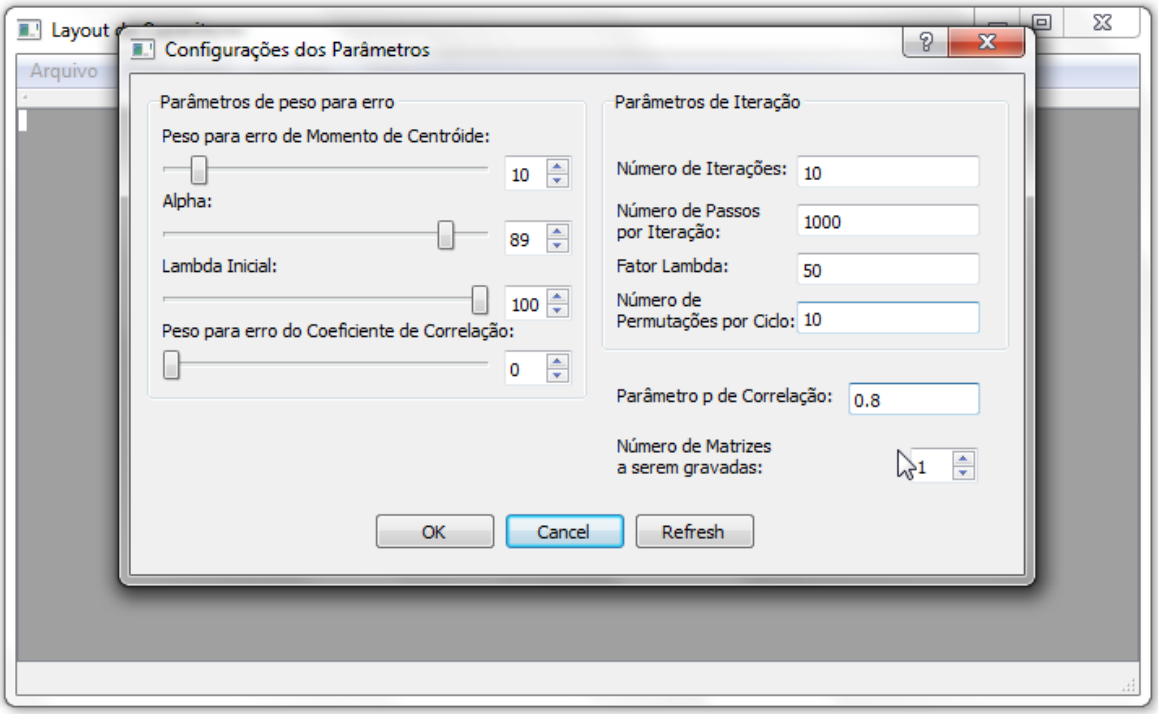


Figura 5.16 – Alterados alguns parâmetros do lado direito também, basta clicar sobre cada uma das caixas de texto.

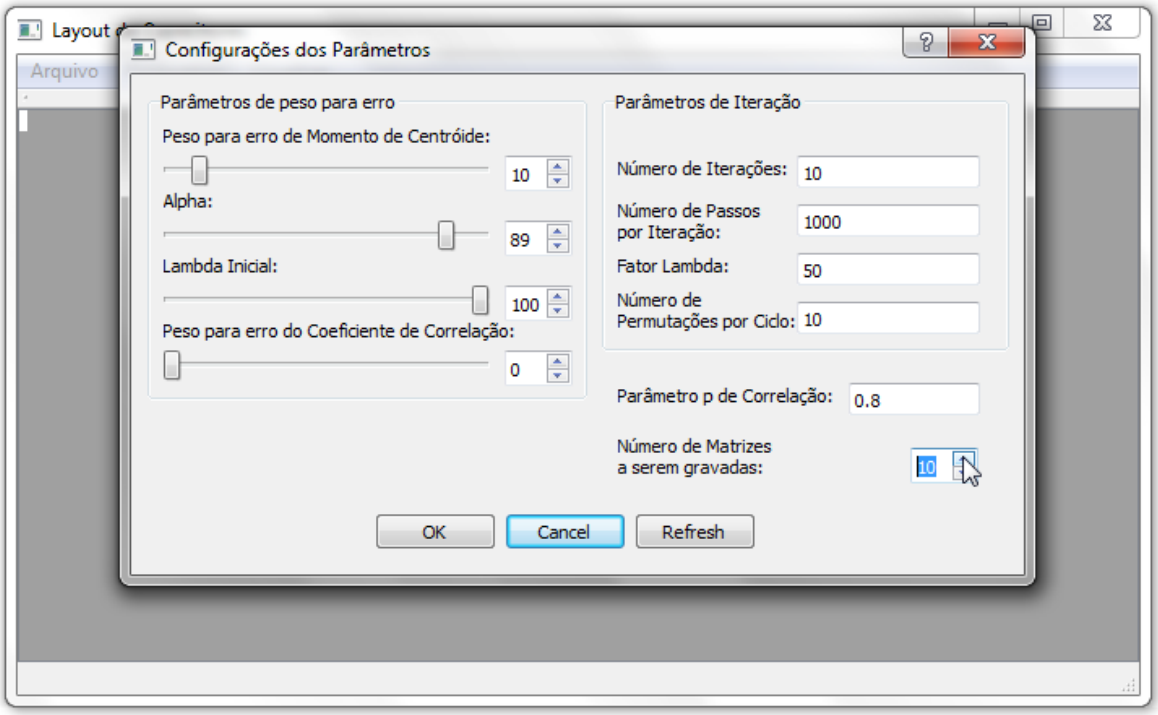


Figura 5.17 – O valor máximo do número de matrizes a serem gravadas é delimitado pelo número de iterações.

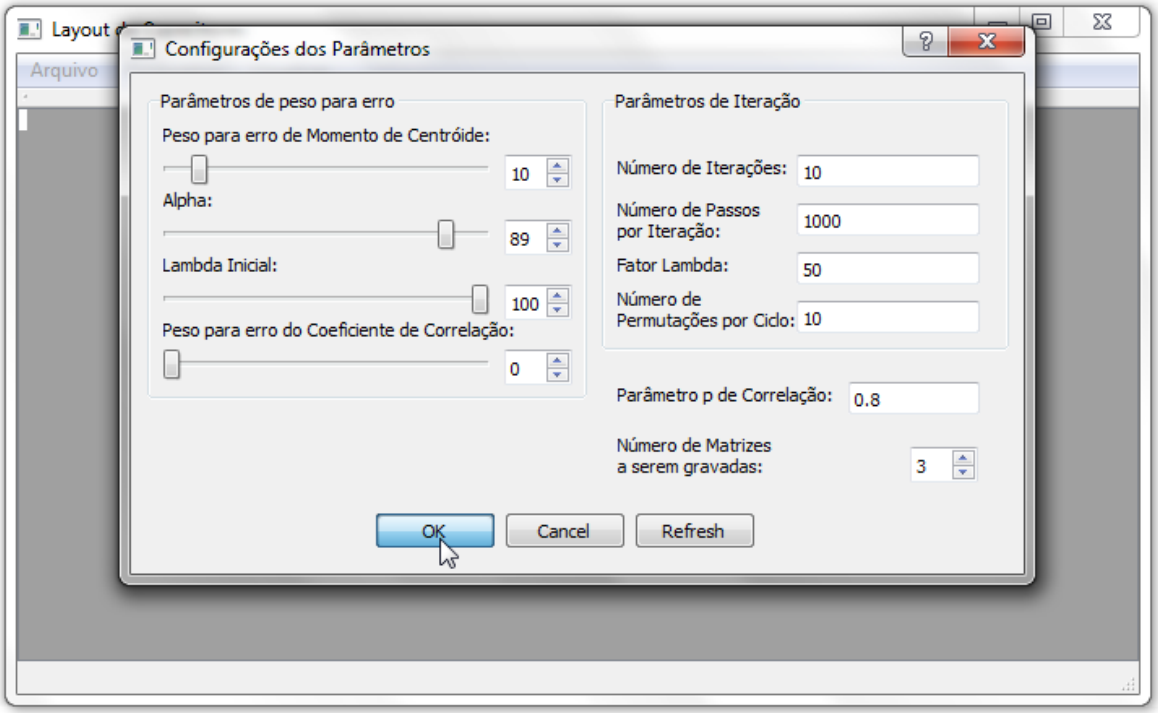


Figura 5.18 – Configurado o número de matrizes para 3, por exemplo. Para finalizar, clica-se em Ok. Após isso, o *software* retorna ao menu inicial.

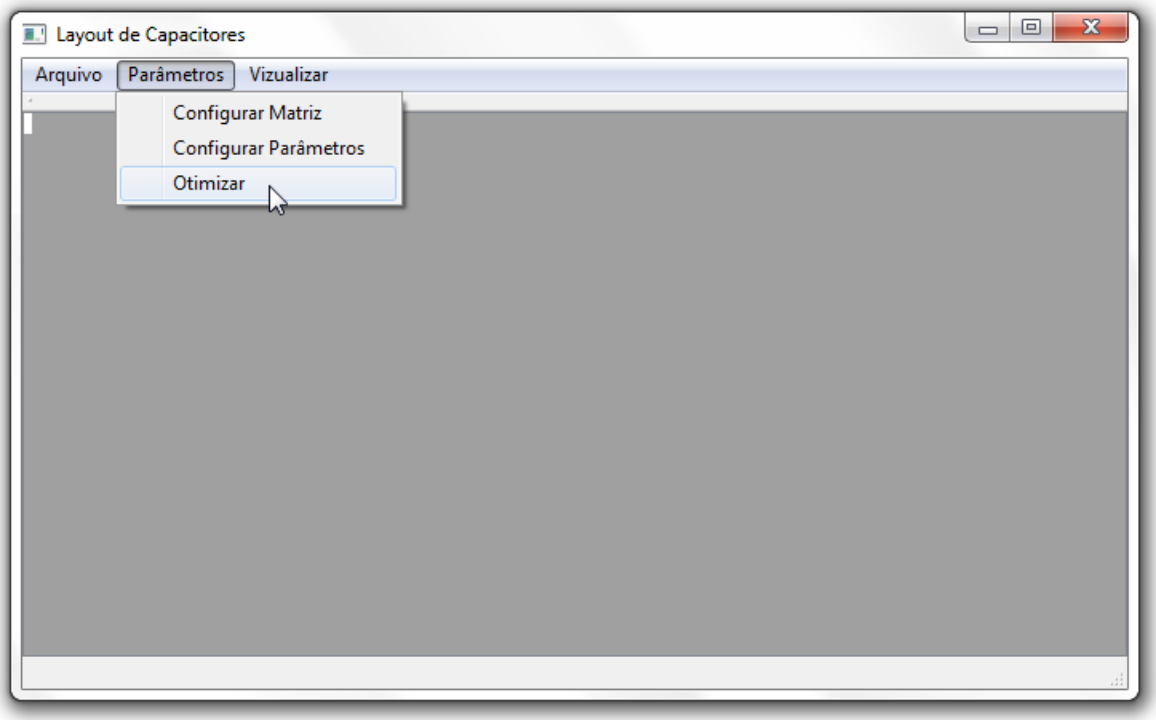


Figura 5.19 – Após clicar em **Otimizar**, começa o processo de otimização.

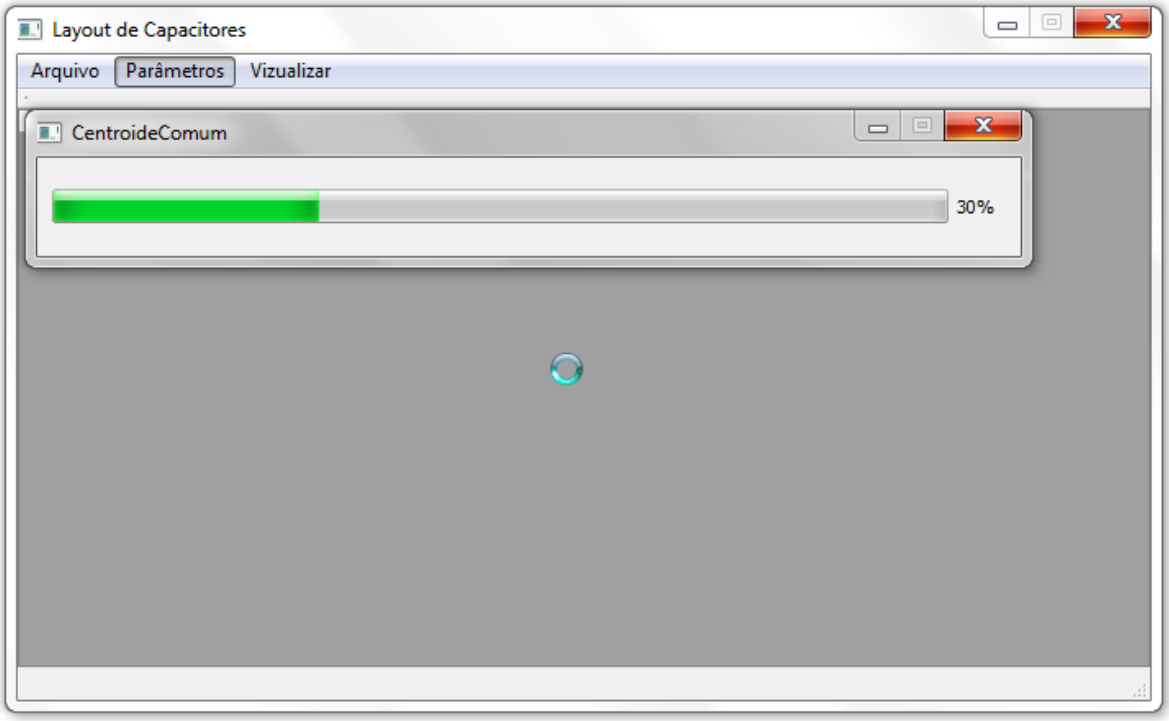


Figura 5.20 – Otimização progredindo, a barra de progresso mostra o quanto já foi feito.

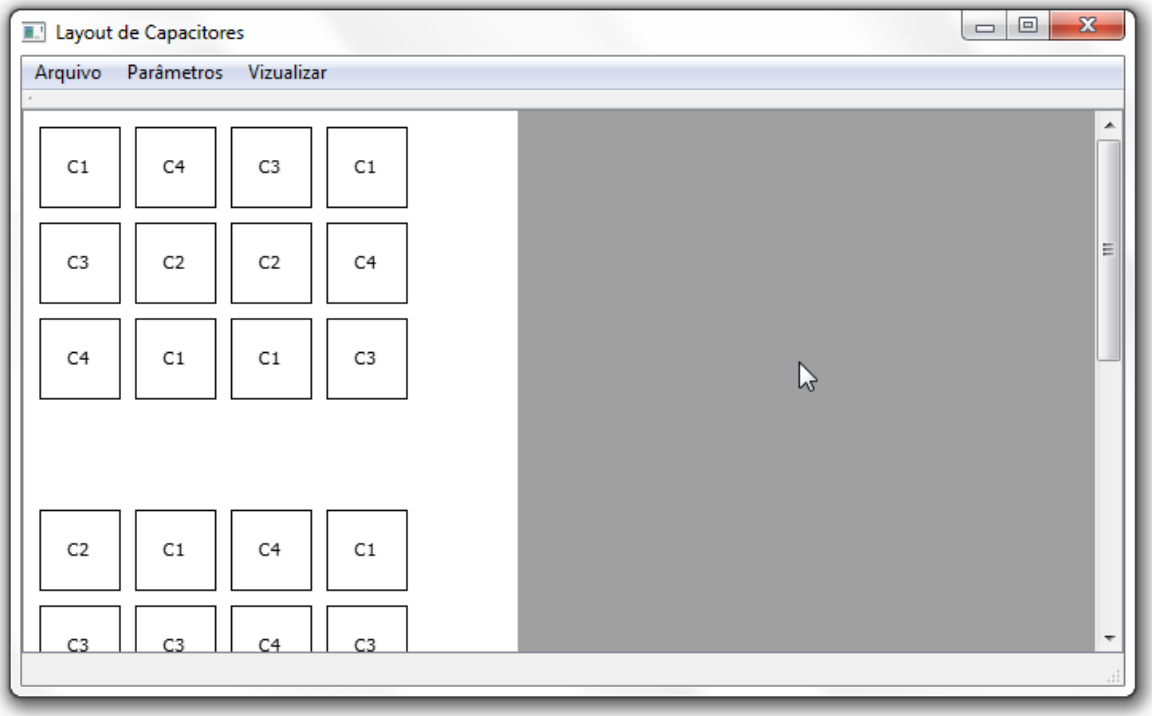


Figura 5.20 – Resultados.

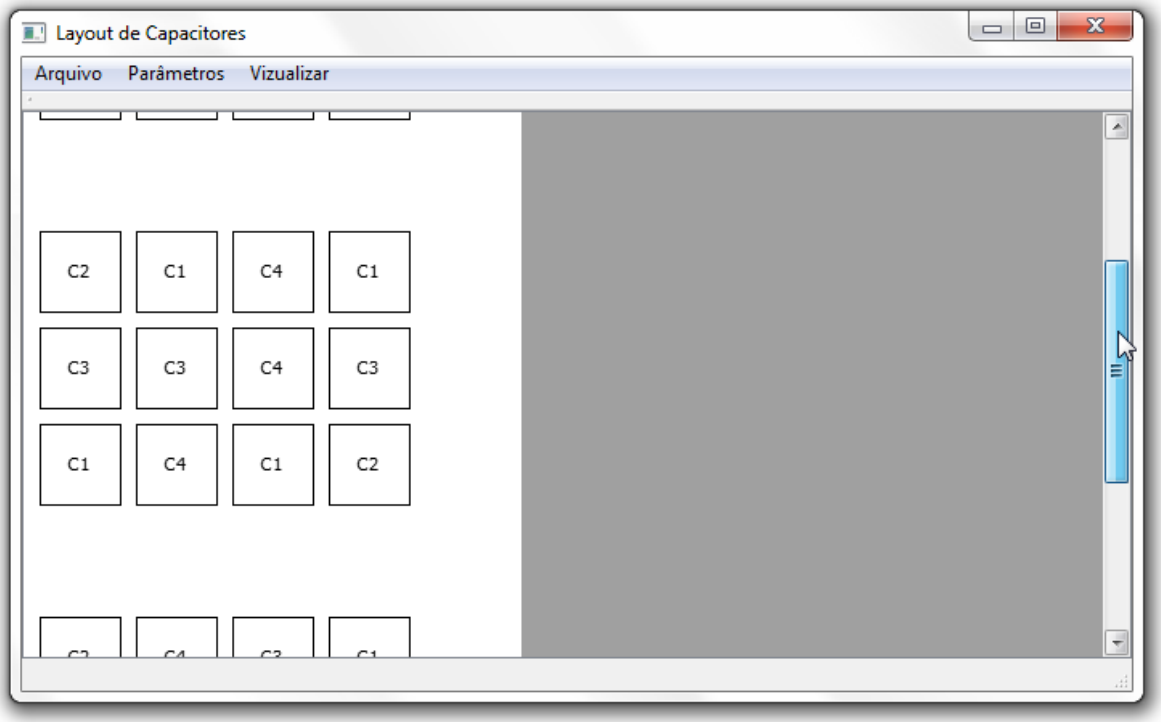


Figura 5.21 – Descendo a barra de rolagem, é possível ver os outros resultados.

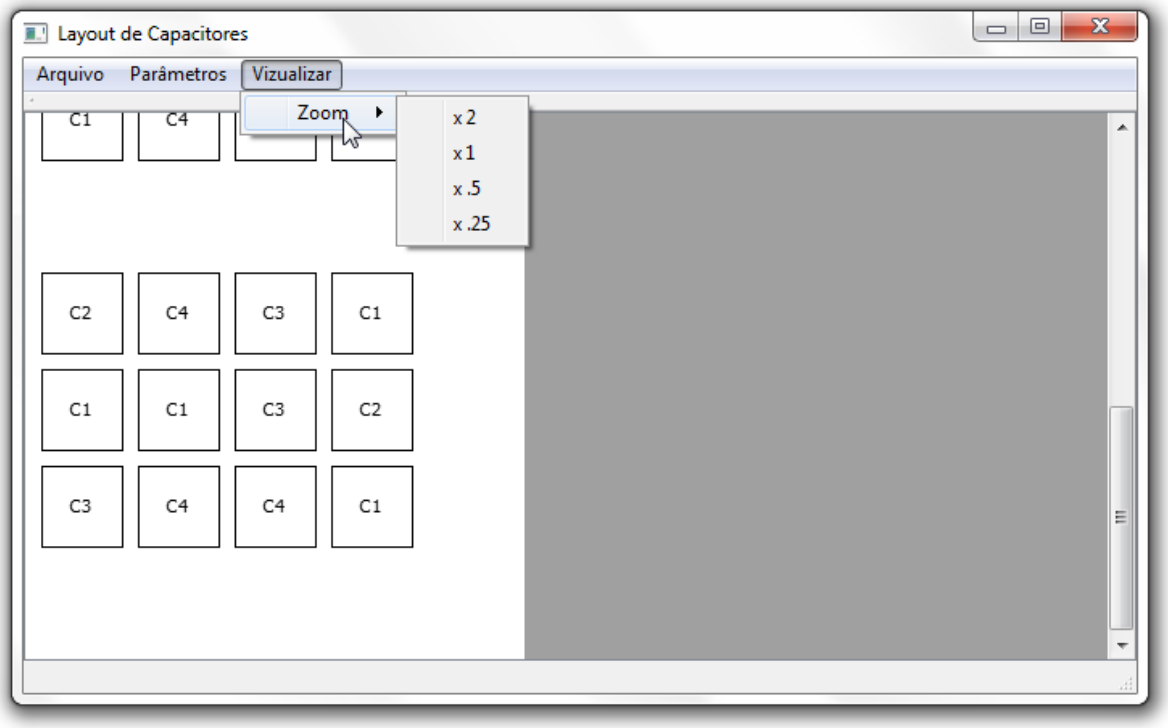


Figura 5.22 – Ajustando o *Zoom*, para que seja possível ver todas as matrizes.

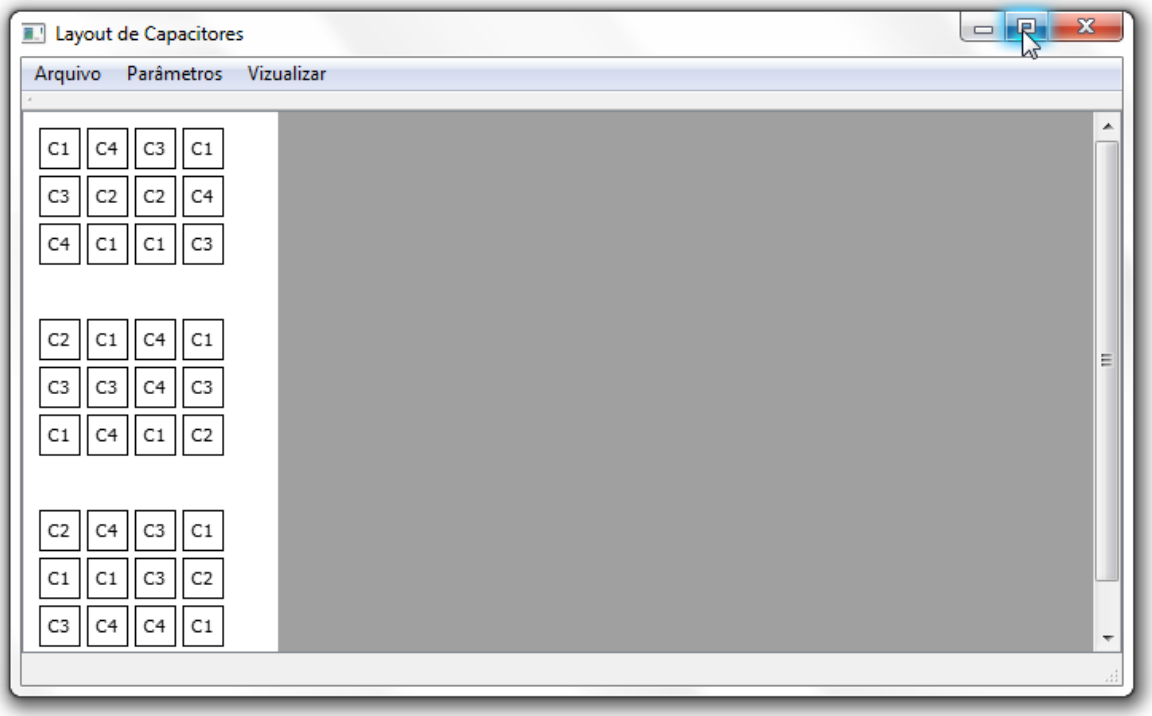


Figura 5.23 – Maximização da janela.

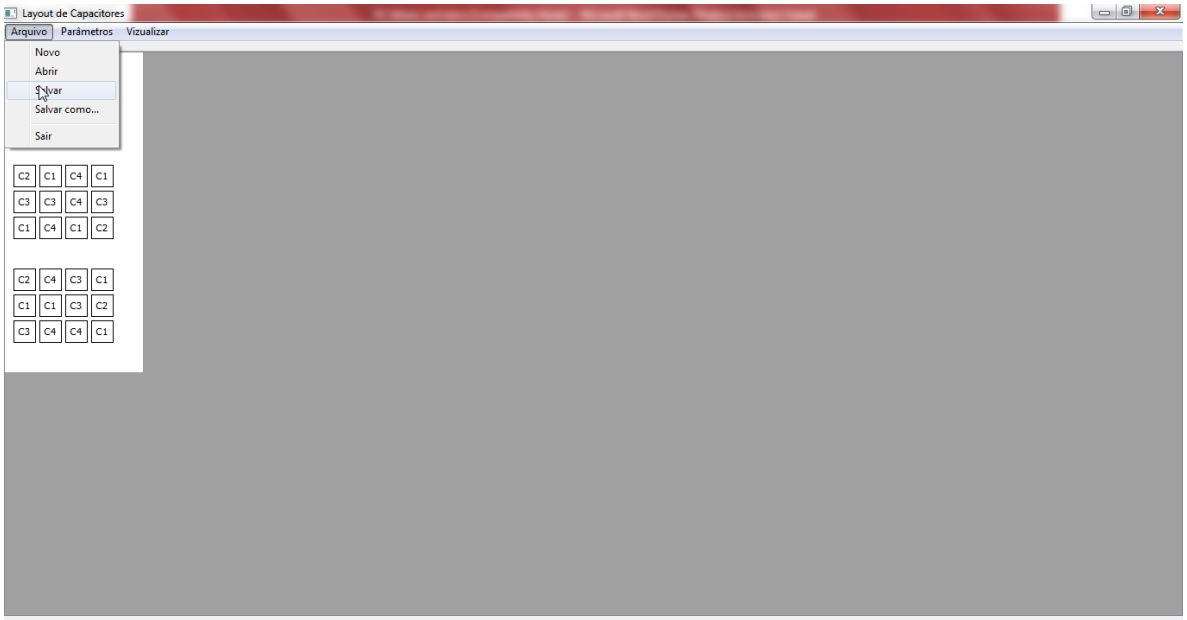


Figura 5.24 – Em **Arquivo→ Salvar** ou em **Arquivo→ Salvar Como** é possível salvar os resultados obtidos. Clicando em **Salvar**, o nome gerado é dado pela data e horário do sistema. Para a data 03/03/2013 e horário 15:15:15, o nome padrão do arquivo seria "Layout 03.03.2013 15.15.15.txt".

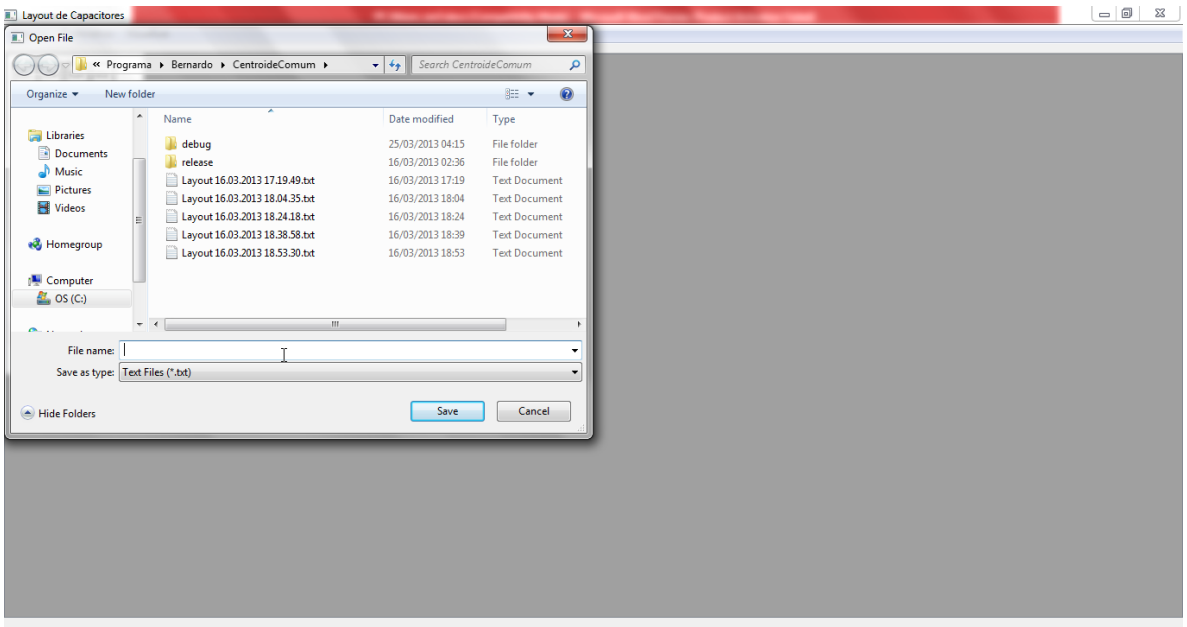


Figura 5.25 – Clicando em **Salvar Como**, surge a janela mostrada.

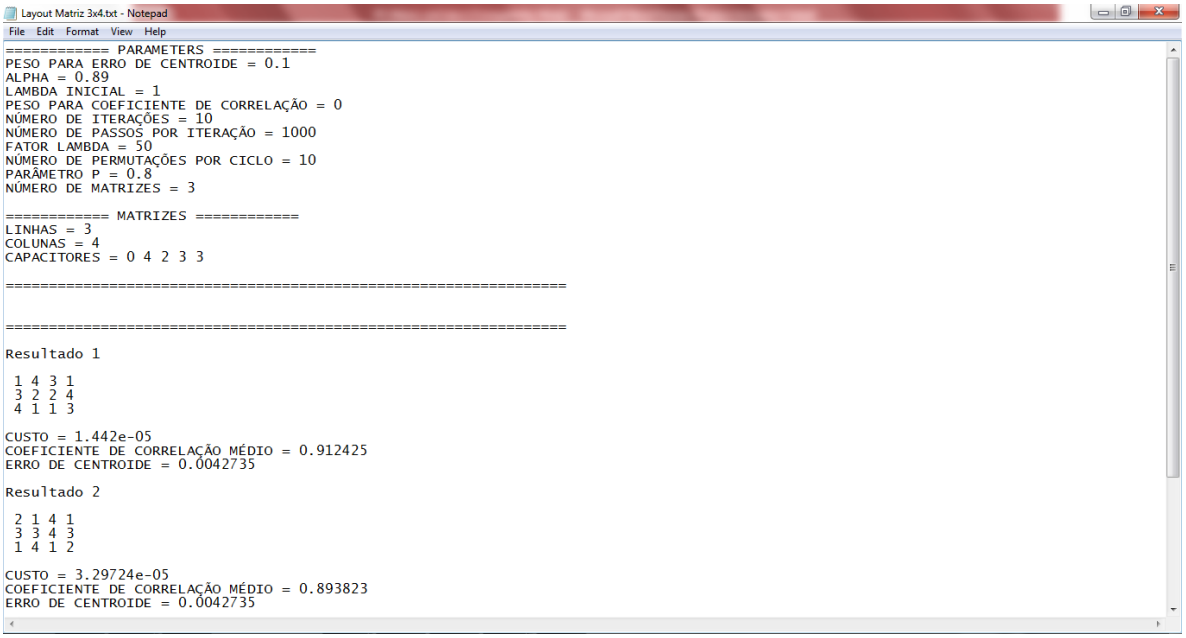


Figura 5.26 – Salvando e abrindo o arquivo texto, pode-se ver os dados referentes à simulação. Caso houvesse algum capacitor *dummy*, ele seria representado pelo número 0 nos resultados.

# Capítulo 6

# Conclusões e Trabalhos Futuros

## 6.1 – Conclusões Sobre o Trabalho

De acordo com os resultados obtidos no Capítulo 5, o programa está funcionando de maneira correta e os resultados gerados por ele são satisfatórios.

A primeira conclusão é que o objetivo do programa então, foi cumprido, pois, com ele, é possível gerar *layouts* com erro de centroide nulo de forma automática, além de utilizar a média dos coeficientes de correlação para melhorá-lo ainda mais, economizando tempo que poderia ser despendido pelo projetista (até mesmo sem que os próprios fossem encontrados e matrizes maiores fossem utilizadas para que as soluções fossem mais fáceis), quando não é possível anular o erro de centroide comum, mesmo assim o programa logra êxito em otimizar a matriz. A Figura 5.26 é um exemplo em que o coeficiente de correlação médio entre os componentes foi utilizado para escolher a melhor configuração, apesar de, nesse exemplo, o erro de centroide comum não ter sido nulo.

Apesar disso, a maximização do coeficiente de correlação médio não foi testada exaustivamente, pois, para um dado valor de $\rho$, só é possível saber o valor máximo da média dos coeficientes de correlação testando todas as configurações de matrizes possíveis para suas dimensões e vetor de unitários. O que só é viável para matrizes menores.

É importante ressaltar que os *layouts* gerados pelo *software* podem ser utilizados em diversas aplicações, conforme descrito ao longo do trabalho, como por exemplo, mas não restrito a: Filtros a Capacitores Chaveados, conversores A/D, conversores D/A.

A segunda conclusão é que o trabalho desenvolvido proporcionou a chance de estudar sobre assuntos não vistos em sala de aula, mas que são interessantes e possuem aplicações práticas. Algoritmos estocásticos, como o *Simulated Annealing* e técnicas para *layout* foram duas delas.

Apesar de não ter sido incluído aqui, foram feitos alguns testes para matrizes e constatou-se que para matrizes pequenas (2x2, 3x3) a chance de encontrar matrizes ótimas utilizando os valores padrões para os parâmetros é bem alta. Na medida em que aumenta-se o tamanho da matriz, alguns parâmetros precisarão ser reconfigurados. Em geral, para essas matrizes, o aumento do número de passos por iterações, o fator de resfriamento *alpha* e o número de iterações foram suficientes.

## 6.2 –Trabalhos Futuros

Apesar do que foi desenvolvido até aqui, bastantes funcionalidades ainda poderiam ser implementadas no *software*.

A média dos coeficientes de correlação muitas vezes não pode ser otimizado por não conhecermos o parâmetro do qual ele é dependente, por isso, seria interessante mais

uma figura de mérito a ser otimizada. A facilidade de roteamento dos componentes é uma delas, assim, a partir das matrizes sem erro de centroide, poderia ser escolhida pelo projetista a com roteamento mais fácil.

Outra funcionalidade interessante seria aproximar as razões de capacitores. Antes de definir os capacitores unitários utilizados no filtro, é necessário aproximar as razões de capacitâncias por uma razões de inteiros, porém, supondo um número máximo de unitários, nem sempre a melhor aproximação para a resposta em frequência do filtro consiste em aproximar cada coeficiente de modo individual, então, este cálculo também poderia ser realizado pela ferramenta de modo a automatizar ainda mais o processo de desenvolvimento de determinados CI’s e facilitar mais o trabalho do projetista analógico.

# Bibliografia


[1] JOHNS, D., MARTIN, K., *Analog Integrated Circuit Design*. John Wiley & Sons, 1997.

[2] GREGORIAN, R., TEMES, G. C., *Analog MOS Integrated Circuits for Signal Processing*. John Wiley & Sons, 1986.

[3] KANEKO, M., MASUDA, M., HAYASHI, T., "A novel capacitor placement strategy in ASCCOT: automatic layouter for switched-capacitor circuits". In: *Proc. of the* IEEE *International Symposium on Circuits and Systems* - ISCAS, pp. 2094–2097, May 1993.

[4] LUO, P., CHEN, J., WEY, C., CHENG, L., CHEN, J., WU, W. , "Impact of Capacitance Correlation on Yield Enhancement". In: IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, No. 11, November 2008.

[5] PELGROM, M. J. M., DUINMAIJER, A. C. J., WELBERS, A. P. G., "Matching properties of MOS transistors", IEEE *Journal of Solid-State Circuits*, Vol. 24, No. 5, pp. 1433–1440, October 1989.

[6] KIRKPATRICK, S., JR., C. D. G., VECCHI, M. P., "Optimization by simulated annealing", Science, Vol. 220, pp. 671–680, May 1983.

[7] HOSTICKA, B. J., BRODERSEN, R. W., GRAY, P. R., "MOS Sampled Data Recursive Filters Using Switched Capacitors Integrators". In IEEE *Journal of Solid-State Circuits*, Vol. SC-12, No. 6, December 1977.

[8] LIU, M., *Demystifying Switched Capacitors*, Elsevier, 2006.

[9] Ki, W-H., Temes, G. C., "Optimal Capacitance Assignment of Switched-Capacitor Biquads". In IEEE *Trans. on Circuits & Systems I: Fundamental Theory and Applications*, Vol. CAS-42, No. 6, pp. 334–342, June 1995.

[10] SOARES, C. F. T., *Métodos para Aprimorar o Projeto e o Layout de Filtros Analógicos em Circuitos Integrados CMOS*. D.Sc. dissertation, Universidade Federal do Rio de Janeiro, Janeiro 2009.

[11] SOARES, C. F. T., *Filtros a Capacitores Chaveados CMOS 0.35 μm para a Detecção de Efeito de Cavitação em Turbinas de Usinas Hidroelétricas*. M.Sc. dissertation, Universidade Federal do Rio de Janeiro, Fevereiro 2006.

[12] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E., "Equation of State Calculation by Fast Computing Machines". In *The Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092, June 1953.

[13] STUART, A., ORD, J. K., *Kendall's Advanced Theory Of Statistics*. Orford Univ. Press, 1987, pp. 320-325.