

UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE FÍSICA DE SÃO CARLOS

DEPARTAMENTO DE FÍSICA E INFORMÁTICA

**ANÁLISE DE PROCESSADORES
PARA APLICAÇÃO EM TEMPO
REAL DO MÉTODO DO GRADIENTE
CONJUGADO NO CONTROLE
ÓTIMO DE PROCESSOS**

Benedito Renê Fischer

Tese apresentada ao Instituto de Física
da USP de São Carlos para a obtenção
do título de Doutor em Ciências Física
Aplicada.

USP / IFQSC / SBI



8-2-001065

OK

Orientador:

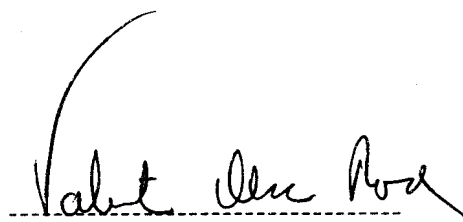
Prof. Dr. Valentin Obac Roda.

São Carlos, 1995.




MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE **BENEDITO RENE FISCHER** APRESENTADA AO
INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 10/01/1995

COMISSÃO JULGADORA:



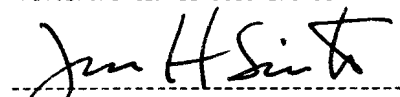
Prof. Dr. Valentin Obae Roda



Prof. Dr. Jan Frans Willem Slaets



Prof. Dr. Newton Geraldo Bretas



Prof. Dr. José Hiroki Saito



Prof. Dr. Noritsuna Furuya

Ao meu filho,

Isaac

"Até aqui ...

nos ajudou o Senhor"

1 Samuel 7:12

AGRADECIMENTOS

O autor expressa seus mais profundos agradecimentos ao seu orientador, Prof.Dr. Valentim Obac Roda, pelo incentivo, confiança e paciência para que este trabalho fosse realizado.

Agradecimentos, também, a todos que contribuíram direta ou indiretamente para a conclusão deste trabalho.

E, como não poderia deixar de fazer, agradecimentos a DEUS, pela Sua fortaleza e grande proteção.

RESUMO

O método do gradiente conjugado, uma das técnicas de otimização irrestrita da Programação Matemática, normalmente é empregado na solução de problemas de controle ótimo, apresentando a vantagem, sobre outros métodos, de convergência razoavelmente rápida e economia de memória. A limitação do método é que o mesmo é calculado para problemas com tempo final fixo.

Neste trabalho é feita uma generalização do método do gradiente conjugado para que ele possa ser empregado na solução de problemas de controle ótimo em tempo real. Através de simulações, foi determinada uma correlação entre a constante de tempo dominante do sistema e o melhor intervalo de operação para o método. Foi proposto um algoritmo original para controle ótimo com tempo final livre, e também estabelecidas as condições sobre a velocidade do processador, para que esse algoritmo possa ser aplicado em tempo real.

Devido ao interesse do controle em tempo real, a velocidade de processamento, o baixo custo do equipamento e as pequenas dimensões físicas são fundamentais. Desta forma, é feita uma análise de desempenho de vários tipos de processadores, com ênfase nas arquiteturas 80x86.

De forma a obter maior velocidade de processamento, foram analisadas implementações de arquiteturas paralelas usando transputadores, redes de computadores e uma arquitetura onde cada processador compartilha um segmento de sua memória com os demais.

ABSTRACT

The conjugate gradient method, one of the techniques of unconstrained optimization of Mathematical Programming, normally is employed in the solutions of optimal control problems. The conjugate gradient method has as main advantages over other methods a relatively fast convergence and minimal use of memory. The method has one limitation. It can only be computed for problems with fixed final time.

In this work, a generalization of the conjugate gradient method is proposed, allowing to extend the applications of the method to the solutions of problems of optimal real time control. Using simulations, a correlation was determined between the dominant time constant of the system and the best operation time interval for the method. An original algorithm was proposed for optimal control with final free time and also the conditions for the processor's speed were established, so that the algorithm can be employed in real time.

Processing speed, low cost of equipment and small physical dimensions are fundamental for real time control. In this manner, a performance analysis is made of the several processors, with emphasis on architectures of the 80x86.

In order to get faster processing, three implementations of parallel architectures were analyzed, using transputers, computer networks and an architecture where each processor shares a segment of memory with the other.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

1.1. Controle Ótimo em Tempo Real	I-1
1.2. Objetivo do Trabalho	I-5
1.3. Apresentação dos Capítulos	I-6

CAPÍTULO 2

A TEORIA DE CONTROLE ÓTIMO

2.1. Sistemas de Controle	II-1
2.2. Histórico da Teoria de Controle	II-3
2.3. Teoria de Controle Moderno	II-7
2.3.1. Teoria de Controle de Sistemas Lineares ...	II-8
2.3.2. Teoria de Controle Ótimo	II-9
2.3.3. Teoria de Controle de Sistemas de Tempo Discreto	II-12
2.3.4. Controle de Sistemas Estocásticos	II-13
2.3.5. Controle Adaptativo de Sistemas	II-14
2.3.6. Técnicas de Controle Ótimo como Ferramen- tas para a Solução de Problemas em outras Áreas da Teoria de Controle Moderno	II-16
2.4. Técnicas para a Solução do Problema de Controle Ótimo	II-17
2.5. Programação Matemática	II-19
2.5.1. Ramificações da Programação Matemática ...	II-21
2.6. Métodos de Otimização Irrestrita	II-24
2.6.1. Método do Gradiente Ótimo	II-26
2.6.2. Método das Direções Conjugadas	II-28
2.6.3. O Método do Gradiente Conjugado	II-32

CAPÍTULO 3

GENERALIZAÇÃO DO MÉTODO DO GRADIENTE CONJUGADO PARA A SOLUÇÃO DE PROBLEMAS DE CONTROLE ÓTIMO

3.1.	Introdução	III-1
3.2.	Generalização do Método do Gradiente Conjugado para o Espaço de Hilbert	III-2
3.2.1.	Expressão para o Gradiente de um Funcional	III-3
3.2.2.	O Gradiente do Funcional de Custo para Sistemas de Controle	III-4
3.3.	O Problema de Teste	III-8
3.3.1.	O Gradiente no Espaço de Funções Contínuas	III-8
3.3.2.	O Modelamento do Motor DC	III-13
3.4.	O Procedimento Computacional	III-18
3.4.1.	Implementação do Algoritmo do Gradiente Conjugado	III-19
3.4.2.	Algoritmo da Busca Unidimensional	III-21
3.4.3.	Resultados Numéricos	III-24
3.4.4.	Análise de Sensitividade	III-28

CAPÍTULO 4

EXTENSÃO DO MÉTODO DO GRADIENTE CONJUGADO PARA CONTROLE ÓTIMO EM TEMPO REAL

4.1.	Introdução	IV-1
4.2.	Sistemas de controle em Tempo Real	IV-2
4.3.	Considerações sobre o Tempo Final de Sistemas em Tempo Real	IV-7
4.4.	Método do Gradiente conjugado com Dilatação do Intervalo de Tempo de Operação	IV-8
4.4.1.	Análise do Resultado	IV-11
4.4.2.	Constante de Tempo Dominante para o Motor DC	IV-16
4.5.	Extensão do Método do Gradiente Conjugado com o Tempo Final Livre	IV-20
4.5.1.	Análise dos Resultados	IV-22

4.6.	Controle Ótimo em Tempo Real de Processos via Método com o Tempo Final livre	IV-26
4.6.1.	Controle Ótimo em Tempo Real em Malha Aberta	IV-28
4.6.2.	Controle Ótimo em Tempo Real em Malha Fechada	IV-32

CAPÍTULO 5

ANÁLISE DE DESEMPENHO DE PROCESSADORES PARA O CONTROLE ÓTIMO EM TEMPO REAL

5.1.	Introdução	V-1
5.2.	A Linha de Computadores "IBM-PC"	V-4
5.3.	Critérios para Medida de Desempenho	V-11
5.4.	Evolução do Desempenho da Família "IBM-PC"	V-15
5.4.1.	Análise com Índices de Desempenho Padrões .	V-18
5.5.	Análise de Desempenho com o Método do Gradiente Conjugado	V-26
5.5.1.	Conclusão sobre o Desempenho do Método	V-30

CAPÍTULO 6

ARQUITETURAS PARALELAS PARA APLICAÇÕES EM TEMPO REAL

6.1.	Introdução	VI-1
6.2.	Paralelização do Algoritmo do Gradiente Conjugado	VI-2
6.2.1.	Abordagem Computacional	VI-4
6.2.2.	Paralelização da Rotina de Integração	VI-6
6.3.	Implementações da Arquitetura Paralela	VI-10
6.3.1.	Tempo de Processamento X Tempo de Transmissão de Dados	VI-11
6.3.2.	Rede de Transputadores	VI-11
6.3.3.	Solução via Rede de Computadores	VI-18
6.3.4.	Solução via Compartilhamento de Memória ..	VI-26

CAPÍTULO 7

CONCLUSÃO	VII-1
-----------------	-------

REFERÊNCIAS BIBLIOGRÁFICAS	BIB-1
----------------------------------	-------

APÊNDICE A

<i>Espaço de Hilbert</i>	A-1
--------------------------------	-----

APÊNDICE B1

<i>Listagem do Subprograma de Minimização pelo Método do Gradiente Conjugado - Parte independente da Aplicação</i>	B1-1
--	------

APÊNDICE B2

<i>Listagem de um dos Subprogramas de Minimização pelo Método do Gradiente Conjugado - Parte dependente da Aplicação</i>	B2-1
--	------

CAPÍTULO 1

Introdução

1.1. Controle Ótimo em Tempo Real

Controlar um processo significa que se deve atuar sobre uma série de variáveis (variáveis de controle ou de entrada) de forma a fazer com que outro conjunto de variáveis do mesmo processo (variáveis controladas) sigam um perfil desejado.

Um controle é dito ótimo se ele forçar as variáveis controladas a seguir um perfil, que minimiza um dado critério de desempenho durante a operação do sistema.

Antes do desenvolvimento dos computadores modernos e dos microcomputadores, a solução desse problema só podia ser obtida analiticamente, e para casos muito simples, com poucas variáveis. É o caso de sistemas lineares, por exemplo, onde as variáveis controladas devem seguir um perfil que otimize um critério de custo quadrático. Nesse caso, se a quantidade de

variáveis de controle e de variáveis controladas for aumentada, a solução analítica tornar-se-ia inviável, pois os computadores de grande porte da época, empregados para a obtenção da solução numérica do problema, não dispunham de tal capacidade de processamento [ATHANS, 66].

Nessa época, meados da década de 60, o controle só podia ser obtido "fora de linha" (*off-line*). As variáveis de controle, empregadas durante toda a operação do sistema, eram calculadas antes do início de sua operação, em um centro de processamento de dados, geralmente distante do local de operação do sistema.

Os recentes desenvolvimentos tecnológicos, ocorridos nas décadas de 80 e 90, nas áreas de microeletrônica e arquitetura de computadores, levaram ao desenvolvimento dos microprocessadores de última geração, integrando mais de um milhão de transistores em uma única pastilha de circuito integrado. Microprocessadores como os 80486, i860 e o Pentium da Intel, bem como o PowerPC da IBM e Motorola, entre outros, permitem a implementação de microcomputadores, cuja potência e velocidade de processamento superam em muito os computadores da década de 60 e 70. Além disso, a drástica redução nos custos e dimensões dos microprocessadores atuais tornam viável a utilização dos mesmos, de forma dedicada, no controle dos processos. A grande velocidade de processamento obtida com tais processadores permite que se tente empregá-los no controle em tempo real de processos.

No controle de um processo em tempo real, o computa-

dor deve calcular as variáveis de controle durante a operação do sistema, tornando evidente a necessidade de velocidade de processamento. Se o controle requerido deve ser ótimo e em tempo real, as exigências quanto à velocidade são mais severas ainda. Nesse caso, as entradas de controle devem ser calculadas durante a operação do sistema, de forma a levar as variáveis controladas a seguir um perfil que minimize ou maximize um dado critério de desempenho.

A obtenção do controle ótimo em tempo real requer, além de um processador de alto desempenho, técnicas adequadas para a solução numérica do problema.

Uma das técnicas empregadas poderia ser a programação dinâmica, entretanto, os algoritmos de programação dinâmica requerem uma grande capacidade de memória. Os computadores atuais podem satisfazer esta exigência, alcançando, em alguns casos, até 32 Mbytes de memória. Porém, o grande número de escrita e leitura na memória, exigidos pelo método, pode comprometer os requisitos de velocidade.

Outra técnica empregada na solução de problemas de controle ótimo é o método do gradiente conjugado. Diferentemente da programação dinâmica, o método do gradiente conjugado não requer tanta memória de computador, além de converge rapidamente para a solução. Apesar de ter sido proposto em 1952, por Hestenes e Stiefel [HESTENES, 52], para a solução de um sistema de equações lineares, o método do gradiente conjugado continua atual. Presentemente, tem sido pesquisado na solução de sistemas de

equações lineares com um grande número de variáveis, utilizando computação massivamente paralela [CRONE, 94][CRONE, 93][EVANS, 93].

O método do gradiente conjugado, como desenvolvido por Hestenes, não pode ser aplicado diretamente na solução dos problemas de controle ótimo. É necessário fazer a generalização do mesmo para o espaço das entradas de controle (um espaço de *Hilbert* - vide *APÊNDICE A*) do sistema que se deseja controlar. Essa generalização é apresentada neste trabalho, no *capítulo 3*.

Atualmente, na solução de problemas de controle ótimo, sempre que o método do gradiente conjugado é empregado, ele é aplicado a problemas com o tempo final fixo e definido *a priori*. Tal limitação impede sua aplicação ao problema de controle ótimo em tempo real, uma vez que nesses sistemas, na maioria dos casos, o tempo final não é previamente conhecido.

Neste trabalho, através de simulações, foi determinada uma correlação entre a constante de tempo dominante do sistema e o melhor intervalo de operação, para o método do gradiente conjugado. Foi proposto um algoritmo original para o controle ótimo com tempo final livre pelo método do gradiente conjugado. Foram estabelecidas, também, as condições sobre a velocidade do processador para que o algoritmo de controle ótimo com tempo final livre possa ser aplicado em tempo real. Finalmente, foi proposta uma modificação do algoritmo para tempo final livre, através de sucessivas aplicações do método do gradiente conjugado, em um esquema tipo "linha de montagem" ou "canalização" (pi-

peline), modificação esta que abranda as restrições sobre o tempo de processamento em sistemas de tempo real. Esta abordagem permite que processadores mais lentos sejam aplicados com sucesso no controle ótimo em tempo real.

1.2. Objetivo do Trabalho

A grande vantagem do uso do método do gradiente conjugado na solução de problemas de controle ótimo é que o mesmo pode ser aplicado em sistemas não lineares e sujeitos a restrições, tanto nas variáveis de controle como nas de estado. Isto significa que o método do gradiente conjugado pode ser aplicado a qualquer tipo de sistema, desde que o mesmo tenha sido convenientemente expresso por equações de estado.

Apesar dessas vantagens, a limitação do método do gradiente, de somente poder ser aplicado em problemas com tempo final fixo, impede sua aplicação no controle ótimo em tempo real.

O objetivo deste trabalho é justamente estender a aplicação do método do gradiente conjugado para a solução de problemas de controle ótimo em tempo real, bem como investigar a viabilidade de implementação desse controle nos processadores atuais.

Considerando que a principal dificuldade no cálculo do controle ótimo em tempo real é a velocidade de processamento,

foi escolhido o problema de controle ótimo de um motor DC (*Direct Current*) como padrão para a análise do desempenho dos diferentes processadores. Nesse caso, a entrada de controle é a tensão de campo, necessária para o motor seguir uma entrada em degrau de 10 radianos. Como critério de desempenho, foi escolhida a minimização da integral do erro entre a posição real do eixo do motor e a posição desejada de 10 radianos, durante o tempo de operação do sistema.

1.3. Apresentação dos Capítulos

No **capítulo 2**, é feito um histórico da teoria de sistemas de controle, abordando a teoria de controle convencional e a teoria de controle moderno, na qual são apresentadas, em detalhes, as técnicas da teoria de controle ótimo. Ainda nesse capítulo, é introduzido o método de otimização irrestrita via gradiente conjugado, situando-o como um dos vários ramos da teoria de programação matemática.

A generalização do método do gradiente conjugado para a solução dos problemas de controle ótimo é apresentada no **capítulo 3**, onde é mostrada a dedução da expressão para o gradiente de um funcional de custo em sua forma generalizada. É realizado, ainda, o modelamento do motor DC e descrito o procedimento computacional para gerar as entradas de controle ótimas. Os resultados obtidos são comparados com os da literatura, executando-se a análise de sensibilidade do método à variação de parâmetros.

No **capítulo 4**, é feita a extensão do método do gradiente conjugado para a solução dos problemas de controle ótimo em tempo real. Inicialmente é estabelecida a diferença entre sistemas "em linha" (on-line), controle digital direto e sistemas em tempo real. Também é determinada uma correlação entre a constante de tempo dominante do sistema e o intervalo de operação do método do gradiente conjugado. O algoritmo para o controle ótimo com tempo final livre, pelo método do gradiente conjugado, é desenvolvido nesse capítulo. Também são obtidas as condições para a aplicação do algoritmo no controle ótimo em tempo real, bem como sua modificação para uso no esquema de linha de montagem (*pipeline*), esquema este, que permite trabalhar com restrições menos severas no tempo de processamento.

A análise de desempenho dos processadores, com ênfase na família 80x86, é apresentada no **capítulo 5**. Nesse mesmo capítulo, também são definidos alguns índices padrões de desempenho, empregados com frequência na literatura. O desempenho do método do gradiente conjugado é medido nos processadores da família 80x86, fornecendo indicações para a sua implementação no controle em tempo real.

Para obter maior velocidade de processamento, é apresentada no **capítulo 6** a análise de três arquiteturas: com transputadores; com redes de computadores; e com processadores que trocam dados através do compartilhamento de posições comuns de suas memórias.

A conclusão do trabalho é apresentada no **capítulo 7**.

CAPÍTULO 2

A Teoria de Controle Ótimo

2.1. Sistemas de Controle

Os sistemas de controle automático estão de tal maneira difundidos na vida cotidiana que sua existência nem sempre é notada. Quando se fala em controle automático no ambiente doméstico, pensa-se logo em sistemas específicos, como o controle remoto do televisor ou o da garagem. Entretanto, outros sistemas, que do ponto de vista da teoria de controle de processos são até mais elaborados que estes, acabam passando despercebidos. É o caso das lavadoras de roupa, de louças e até dos sistemas mais simples, com termostato, como o aquecedor do aquário, o forno elétrico e o ferro de passar roupas.

Se no ambiente doméstico os sistemas de controle automático são encontrados com tal profusão, é impossível de se imaginar um sistema de manufatura que não os empregue. Nessa área, no entanto, as restrições de precisão, confiabilidade e

disponibilidade são muito mais severas que no caso doméstico, pois podem, em certos casos, até implicar em risco de vida para os operadores.

Considerando casos ainda mais severos, como o controle e posicionamento de sondas e naves espaciais, pode-se ter uma idéia do grau de sofisticação e desenvolvimento da teoria de controle moderno de processos.

Um sistema de controle é dito automático se, em menor ou maior grau, dispensar a participação do homem durante sua operação. A análise e projeto de tais tipos de sistemas despertaram grande interesse, desde a mais remota antigüidade, devido à necessidade de eliminar a participação do homem em tarefas que são cansativas, rotineiras e até mesmo perigosas.

Os sistemas de controle automático de processos podem ser classificados em dois grandes grupos: sistemas com malha aberta e sistemas com malha fechada (realimentados).

Nos sistemas com malha aberta, a grandeza a ser controlada (variável de saída do processo) não tem influência sobre o ciclo de controle do processo. O controle da máquina de lavar roupas é um exemplo deste tipo de sistema, pois a variável de saída, grau de limpeza da roupa, não influencia o ciclo de trabalho da máquina: ela vai gastar sempre o mesmo tempo, quer seja alimentada com roupa limpa ou com roupa suja.

Nos sistemas com malha fechada, a variável de saída

do processo é medida e comparada com um valor de referência, de modo a gerar um sinal de erro. Este sinal é empregado para atuar no processo de forma a manter a saída o mais próximo possível do valor desejado. Sistemas com termostato, como o do aquário ou o do ferro de passar roupas elétrico, constituem exemplos simples deste tipo de sistema, nos quais o aquecimento é desligado quando a temperatura atinge o valor desejado. Outro exemplo, ainda mais simples e mais ilustrativo, é o sistema de bóias dos depósitos de água. Neste caso, enquanto não se atinge o nível de referência, a bóia mantém a entrada de água aberta. O controle é realizado de maneira proporcional ao sinal de erro, ou seja, quando o nível do reservatório estiver próximo do nível desejado (cheio), a válvula estará quase fechada; à medida que o nível de água abaixa, a válvula se abre, permitindo maior passagem de água para encher o depósito (voltar ao nível de referência).

2.2. Histórico da Teoria de Controle

Um dos primeiros sistemas de controle de malha aberta que se tem conhecimento foi o dispositivo de Hero, na época da Grécia antiga, para abrir automaticamente as portas de um templo.

Outro exemplo de sistema de controle sem realimentação é o de cartões perfurados, inventados na França em 1801 por Joseph Jacquard para o controle de teares.

Já na Roma antiga, os engenheiros romanos usavam um

sistema de válvulas flutuantes para controlar o nível da água em seus aquadutos, constituindo um sistema de controle em malha fechada.

Em 1788, James Watt desenvolveu o regulador de esferas, para controle de velocidade de sua máquina a vapor, e este é considerado um dos primeiros sistemas de controle em malha fechada a ser utilizado em larga escala [D'AZZO,75].

Em 1868, Maxwell fez um estudo analítico da estabilidade deste regulador [BELLMAN,64]. Este trabalho foi complementado posteriormente, em 1919, pelo engenheiro russo Wischnegradsky, que considerou um regulador de esferas de terceira ordem [D'AZZO, 75].

Minorky, em 1922, publicou um trabalho envolvendo uma das primeiras aplicações da introdução deliberada de elementos não lineares em sistemas de malha fechada, em um estudo da pilotagem automática de navios, aplicada no navio de guerra americano "Novo México" [D'AZZO, 75].

Em 1934, apareceu o primeiro trabalho sobre a teoria de servomecanismos , publicado por Hazen no *Journal of the Franklin Institute*, sendo que o termo *servomecanismo* foi originado nesse artigo, como proveniente dos vocábulos servo (escravo) e mecanismo.

Neste mesmo ano surgiu um trabalho sobre amplificadores realimentados, de grande importância para a teoria de contro-

le, desenvolvido por Black [D'AZZO,75] e que complementava um trabalho anterior de Nyquist [BELLMAN,64]. Seguiram-se a este os trabalhos de Bode, Hall e Harris [BELLMAN,64] [D'AZZO,75], que até o final dos anos 50 vieram a se constituir nas técnicas de domínio freqüencial da chamada teoria de controle convencional.

Nesta fase do desenvolvimento da teoria, os conceitos desenvolvidos em amplificadores realimentados foram usados também no projeto de servomecanismos no domínio da freqüência. Em particular, os conceitos de resposta em freqüência, largura de banda e margens de ganho e de fase foram empregados mais ou menos por tentativa e erro.

Outra abordagem na teoria de controle convencional, que também se iniciou após o fim da segunda guerra e começo dos anos 50, são as técnicas de domínio do tempo. Neste caso, são empregados critérios como tempo de subida, tempo de acomodação e sobre-elevação.

Em 1948, Evans introduziu o método do lugar das raízes, que se revelou uma excelente ferramenta para o projeto de sistemas de controle com realimentação, sendo também uma ponte entre os métodos de domínio do tempo e os de domínio da freqüência [ATHANS, 66].

Os métodos desenvolvidos até então eram úteis na análise e projeto de servomecanismos lineares, porém não podiam ser aplicados nos sistemas não lineares, principalmente nos sistemas de controle com relês, de bastante interesse na época, por repre-

sentarem um tipo de amplificador de potência simples e de baixo custo.

Duas novas abordagens foram posteriormente desenvolvidas para abranger o caso de sistemas não lineares, como aqueles com relês: o método da função descritiva, que permite a análise de estabilidade de um sistema não linear em malha fechada no domínio das frequências, e o método do espaço de fase, que permite o projeto de um sistema de controle não linear no domínio do tempo.

A abordagem estatística constante na última fase de desenvolvimento da teoria clássica de controle de processos, tendo, como critério de qualidade, a minimização do *erro quadrático médio*, iniciou uma nova tendência a ser seguida, posteriormente, pela teoria de controle moderno: a otimização de um critério de custo. A teoria foi proposta de maneira independente pelo norte-americano Norbert Wiener e pelo russo A. N. Kolmogorov em 1942, sendo, em 1950, estendida por Zadeh e Ragazzini.

Desta forma, a teoria convencional de controle de processos, ou teoria clássica de controle de processos, é caracterizada por técnicas como análise de estabilidade, resposta em frequência, lugar das raízes, plano de fase e funções descritivas. Tal teoria, entretanto, pode ser aplicada a uma classe muito restrita de problemas na análise e projeto de sistemas de controle. Esta classe envolve os sistemas com uma variável de entrada e uma de saída (sistemas monovariáveis), invariantes no tempo e sem vínculos nas variáveis do processo.

Outra desvantagem da teoria clássica é que ela geralmente é um processo de *tentativa e erro*, onde as técnicas anteriormente citadas são usadas iterativamente para definir os parâmetros que seriam aceitáveis para um sistema. Os critérios seriam definidos nos domínios do tempo e de frequência, definindo parâmetros "aceitáveis" para o tempo de acomodação, sobre-elevação, margens de ganho e de fase e largura de banda.

No final da década de 50, começou a ser desenvolvida uma nova teoria de controle, que foi chamada de teoria de controle moderno. Esta teoria pode ser aplicada a sistemas mais gerais, compostos com múltiplas entradas e/ou múltiplas saídas. Os primeiros trabalhos tratavam dos casos de sistemas lineares com múltiplas entradas e saídas e com parâmetros invariantes no tempo. Na verdade, nos casos mais gerais, as equações de estado podem ser não lineares, com coeficientes variáveis no tempo; as entradas e saídas podem estar sujeitas a restrições de saturação e as variáveis podem ser ainda não-determinísticas (estocásticas).

2.3. Teoria de Controle Moderno

A teoria de controle moderno emprega um conjunto de equações diferenciais (equações de estado) para descrever a dinâmica do sistema a ser controlado. Há várias ramificações desta teoria, dependendo da forma com que for modelado matematicamente o sistema real e dos parâmetros que se deseja contro-

lar. Algumas dessas ramificações serão descritas resumidamente nas 5 subseções que seguem, a saber: teoria de controle de sistemas lineares, teoria de controle ótimo, teoria de controle de sistemas de tempo discreto, controle de sistemas estocásticos e controle adaptativo de sistemas.

2.3.1. Teoria de Controle de Sistemas Lineares

Quando o sistema pode ser descrito por equações diferenciais lineares de primeira ordem com coeficientes constantes (sistema invariante no tempo), as equações de estado são dadas por:

$$\dot{\mathbf{x}}(t) = \mathbf{Ax}(t) + \mathbf{Bu}(t) \quad \mathbf{x}(t_0) = \mathbf{x}_0 \quad (2.3.1-1)$$

$$\mathbf{y}(t) = \mathbf{Cx} + \mathbf{Du}(t) \quad (2.3.1-2)$$

onde

\mathbf{x} é um vetor de $(n \times 1)$ componentes, chamado de *vetor de variáveis de estado* do sistema, ou simplesmente *variável de estado*;

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt};$$

\mathbf{u} é um vetor de $(m \times 1)$ componentes, chamado de *vetor de variáveis de controle* ou *variável de entrada* ou *variável de controle*;

A é a matriz de $(n \times n)$ componentes relacionados com os parâmetros do sistema;

B é uma matriz de $(n \times m)$ componentes, relacionada com os parâmetros de entrada do sistema;

y é o vetor de $(p \times 1)$ componentes das variáveis de saída do sistema;

C é uma matriz de $(p \times n)$ e **D** é uma matriz de $(p \times m)$, relacionadas com os parâmetros de saída do sistema.

Em sistemas descritos por equações de estado como em (2.3.1-1) e (2.3.1-2), as ferramentas da álgebra linear e análise matricial podem ser empregadas, consistindo na chamada *teoria de sistemas lineares multivariáveis* [CHEN, 70]. Tal teoria pode ser aplicada apenas para a análise e projeto de sistemas de controle lineares, a exemplo da teoria clássica, porém com múltiplas variáveis de entrada e de saída.

A teoria de sistemas lineares permite a determinação da estabilidade, projeto de compensadores, determinação da controlabilidade e observabilidade nos sistemas multivariáveis.

2.3.2. Teoria de Controle Ótimo

Um índice de desempenho pode ser associado às equações

de estado de um dado sistema, de forma que o mesmo possa ser operado visando a otimização (maximização e minimização) deste índice. Um controle que opera segundo esta condição é chamado de controle ótimo.

Nos casos mais gerais, a teoria de controle ótimo pode ser aplicada a sistemas não lineares sujeitos a restrições de saturação, tanto nas variáveis de estado como nas de controle; o índice de desempenho ou funcional de custo também pode ser não linear.

No início do desenvolvimento dessa teoria, entretanto, quando se buscava soluções analíticas para o problema, grande parte dos esforços foram concentrados no estudo de sistemas lineares, com índice de desempenho quadrático. Nesse caso, as equações de estado que descrevem o sistema são as equações (2.3.1-1) e (2.3.1-2) da subseção anterior e o índice de desempenho é dado por:

$$ID = \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, t) dt = \int_0^{t_f} [\mathbf{x}^T(t) \mathbf{Q} \mathbf{x}(t) + \mathbf{u}^T(t) \mathbf{Z} \mathbf{u}(t)] dt \quad (2.3.2-1)$$

As primeiras tentativas para a solução de problemas de controle ótimo ocorreram no início dos anos 50, em problemas de tempo ótimo. Nessa época, as provas da existência do ótimo eram baseadas em princípios geométricos e um tanto heurísticos.

No período de 1953 a 1957, os problemas de tempo ótimo foram intensivamente estudados por matemáticos soviéticos e norte-

americanos, como Bellman, Gramkrelidze, Krasovskii e LaSalle. Reconheceu-se, então, que os problemas de controle de tempo ótimo eram essencialmente problemas do cálculo de variações [Athans, 66].

A teoria do cálculo variacional clássica, entretanto, quando aplicada aos problemas de controle, levavam a vínculos muito difíceis de serem resolvidos. Essa dificuldade levou Pontryagin, em conjunto com Boltyanskii e Gamkrelidze, a estabelecer e provar o princípio do máximo em 1958. O princípio do máximo pode ser visto como uma extensão da formulação Hamiltoniana para problemas variacionais [Athans, 66][BELLMAN, 64].

Paralelamente, entre 1953 e 1957, Bellman desenvolveu uma técnica chamada de programação dinâmica, que pode ser empregada com sucesso na solução de problemas de controle ótimo. A programação dinâmica pode ser vista como uma extensão da formulação de Hamilton-Jacobi para problemas variacionais [Bellman, 64].

A programação dinâmica é uma técnica apropriada para o uso de computadores digitais. Esta técnica requer muita memória do computador, mas, considerando o desenvolvimento dos computadores modernos, a programação dinâmica pode ser aplicada atualmente em problemas de maior porte.

Há, entretanto, técnicas mais eficientes e que não requerem tanta memória do computador para a solução dos problemas de controle ótimo, como o método do gradiente conjugado. O emprego destas novas técnicas, associadas ao desenvolvimento de computadores mais rápidos, permite que o controle ótimo possa ser obtido,

mesmo nos casos mais gerais, para sistemas multivariáveis, não lineares e sujeitos a restrições de saturação nas variáveis de controle e de estado, e com critérios de desempenho não lineares.

2.3.3. Teoria de Controle de Sistemas de Tempo Discreto

Quando um computador digital é utilizado para gerar as entradas de controle para um dado sistema, os sinais gerados não mudam de valor continuamente com o tempo, mas sim, em instantes discretos do tempo. Isto se deve à natureza discreta dos computadores digitais, cujas saídas das operações aritméticas mudam de valor em intervalos múltiplos do período de seu relógio interno.

Assim, se o sistema a ser controlado é caracterizado por uma equação diferencial linear a coeficientes constantes e se suas entradas são mantidas constantes por intervalos T de tempo, então sua dinâmica pode ser representada por uma equação linear à diferença. Este tipo de equação é mais fácil de ser resolvida numericamente do que a correspondente equação diferencial. Uma teoria especial, chamada teoria de controle para sistemas de tempo discreto, foi desenvolvida para tratar estes casos [CADZOW, 70] [HOUPIS, 85].

O cálculo do controle ótimo em sistemas de tempo discreto é mais fácil de ser obtido do que no caso de sistemas contínuos. Entretanto, na maioria dos casos da literatura, a teoria fica limitada ao caso de sistemas lineares com custo quadrático.

limitada ao caso de sistemas lineares com custo quadrático.

2.3.4. Controle de Sistemas Estocásticos

Uma das limitações da teoria de controle ótimo determinístico, logo percebida pelos pesquisadores que trabalhavam há mais tempo na área, é que, no caso, não há diferença entre um programa de controle (sistema em malha aberta) e um controle com realimentação (sistema em malha fechada). Outro problema é que a realimentação ótima simplesmente mapeia o espaço de estado no espaço das variáveis de controle, ou seja, não existe dinâmica na realimentação ótima [ÅSTRÖM, 70].

Para se perceber a real diferença entre o controle ótimo em malha fechada e o controle ótimo em malha aberta, basta que se introduza perturbações nas variáveis de controle que não podem ser previstas de maneira determinística. Sendo a perturbação conhecida *a priori* e o sistema governado por equações diferenciais com solução única, o conhecimento das condições iniciais é equivalente ao conhecimento do estado do sistema em um instante arbitrário. Por tal motivo, nesse caso, não há diferença no desempenho ótimo de um sistema em malha fechada e um em malha aberta.

Dessa forma, a teoria de controle estocástico considera sistemas dinâmicos descritos por equações diferenciais ou por equações à diferença, sujeitos a perturbações que são caracterizadas como processos estocásticos. A teoria procura soluções para os problemas de análise, otimização paramétrica e controle ótimo

estocástico, sendo fundamentada pela teoria de predição e filtragem de Wiener e Kolmogorov [BOX, 76][PANDIT, 83] e nos trabalhos de Kalman e Bucy [ÅSTRÖM, 70].

A solução dos problemas de controle estocástico está fundamentada, principalmente, nos conceitos e técnicas da programação dinâmica, que embora possam ser aplicadas a sistemas não lineares, na maioria dos textos da literatura estão voltadas para sistemas lineares e, no máximo, com custo quadrático [ÅSTRÖM, 70][BERTSEKAS, 76][DAVIS, 85].

2.3.5. Controle Adaptativo de Sistemas

Nos anos mais recentes cresceu muito o interesse em controle de sistemas capazes de se adaptarem a mudanças imprevisíveis em seus parâmetros internos ou no meio ambiente. Este tipo de controle é chamado de controle adaptativo de sistemas [LANDAU, 79][ÅSTRÖM, 89].

Sistemas desse tipo despertam o interesse de projetistas, uma vez que, além da adaptação às mudanças ambientais, os sistemas tentariam compensar falhas em seus componentes internos, podendo também se adaptar a condições não previstas no projeto, ou projetadas erroneamente.

O controle adaptativo de sistemas também é empregado quando se tem uma imprecisão muito grande no modelo dinâmico, quer por desconhecimento, por modelagem imprecisa ou quando o uso de um

modelo completo levaria a um sistema com um número muito grande de graus de liberdade. Os braços manipuladores de robôs são um exemplo desse tipo de problema, onde um modelo preciso, que leve em conta as forças de gravidade, centrífuga, inerciais e os atritos internos, iria requerer um modelo tão complexo que seria inviável [CRAIG, 88].

Este ramo da teoria de controle moderno começou a se desenvolver desde a década de 50, devido às necessidades de projeto de sistemas de piloto automático em aeronaves de alto desempenho. Na década de 60, várias contribuições na teoria de controle moderno, como espaço de estado, teoria de estabilidade para sistemas multivariáveis e teoria de controle estocástico, foram importantes para o desenvolvimento do controle adaptativo. Também a programação dinâmica contribuiu para aumentar a compreensão dos sistemas adaptativos [BELLMAN, 65].

No controle adaptativo, a lei de alteração dos parâmetros do sistema é gerada de forma a otimizar um critério de desempenho advindo da comparação entre a saída do sistema real e a de um modelo de referência.

Devido a características intrinsecamente não lineares e à otimização do critério de desempenho, a solução de problemas de controle adaptativo baseia-se, principalmente, nas técnicas de solução de problemas de controle ótimo não linear, como programação dinâmica [BELLMAN, 65] e, mais recentemente, métodos de gradiente [WIDROW, 85].

2.3.6. Técnicas de Controle Ótimo como Ferramentas para a Solução de Problemas em outras Áreas da Teoria de Controle Moderno

A existência de um método para a solução do problema de controle ótimo em sua forma mais geral, onde se tem equações de estado não lineares, sujeitas a restrições de saturação nas variáveis de estado e de controle e funcional de custo não linear, acaba gerando uma ferramenta poderosa para a solução de problemas da teoria de controle moderno. Nos sistemas de controle estocástico, por exemplo, está implícita a minimização de um critério de erro quadrático, que, ao invés de ser tratada pelas técnicas de controle estocástico, pode ser resolvida com este método mais geral de controle ótimo. O mesmo acontece com os sistemas de controle de tempo discreto e controle adaptativo.

Todos estes casos podem ser considerados casos particulares da teoria de controle ótimo, visto que elas envolvem a determinação de uma política de minimização de um critério de desempenho.

O método do gradiente conjugado, usado neste trabalho, e introduzido na *seção 2.6.3*, é uma técnica para a solução do problema do controle ótimo em sua forma mais geral. Dependendo do sistema a ser controlado, esse método se apresenta como o melhor caminho, senão o único.

2.4. Técnicas para Solução do Problema de Controle Ótimo

Para que o problema do controle ótimo possa ser estabelecido em sua forma mais geral, abrangendo os sistemas não lineares, é necessário que as equações de estado também sejam escritas em uma forma mais geral do que a descrita pelas equações (2.3.1-1) e (2.3.1-2), que servem apenas para modelar sistemas lineares multivariáveis.

A descrição da dinâmica do sistema a ser controlado através de equações de estado, constitui o primeiro passo no desenvolvimento da formulação matemática para obtenção dos algoritmos de controle ótimo. Se o sistema é contínuo no tempo, as equações se apresentam como um conjunto de equações diferenciais de primeira ordem do tipo:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \quad (2.4-1)$$

onde

$\mathbf{x}(t)$ é um vetor de $(n \times 1)$ componentes, representando o estado do sistema no tempo t ;

$\mathbf{u}(t)$ é um vetor de $(m \times 1)$ componentes, que especifica a entrada do sistema e

\mathbf{f} é uma função vetorial de $(n \times 1)$ componentes.

Sendo dadas as equações de estado, um conjunto de condições de contorno sobre as variáveis de estado nos instantes de tempo inicial e final e um conjunto de vínculos sobre as variáveis de estado e de controle, o problema de controle ótimo pode ser enunciado como [NOTON, 72] [TABAK, 71] [ATHANS, 66]:

"Determine um controle admissível $\mathbf{u}(t)$, de forma a minimizar (ou maximizar) um dado critério de desempenho ou função de custo".

Sendo conhecidas as condições de contorno sobre as variáveis de estado, $\mathbf{x}(t_0)$ e $S(\mathbf{x}(t_f))$, onde S é chamado de conjunto objetivo, e t_0 e t_f representam o instante inicial (sempre fixado) e o final (não necessariamente conhecido), respectivamente, a função de custo é descrita como um escalar do tipo:

$$J = G(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} F(\mathbf{x}(t), \mathbf{u}(t), t) dt \quad (2.4-2)$$

onde F e G são funções escalares.

Existem várias teorias utilizadas para resolver o problema de controle ótimo. As principais são:

- 1- Cálculo de variações [ATHANS, 66] [TABAK, 71];
- 2- Princípio do Máximo [ATHANS, 66] [TABAK, 71];

- 3- Programação dinâmica [BELLMAN, 62; 64; 65]
[BERTSEKAS, 76][TABAK, 71];
- 4- Quasilinearização [SAGE, 68];
- 5- Embutimento Invariante (*Invariant Imbedding*)
[SAGE, 68];
- 6- Programação Matemática [TABAK, 71].

Os algoritmos que serão utilizados neste trabalho pertencem a um ramo da Programação Matemática. Assim sendo, a próxima seção trata com mais detalhes a Programação Matemática.

2.5. Programação Matemática

A aplicação das técnicas de Programação Matemática na solução de problemas de controle ótimo iniciou-se por volta de 1960 [TABAK, 71]. A Programação Matemática, a exemplo dos outros métodos citados na seção anterior, procura resolver alguns casos de otimização. Tais problemas envolvem a obtenção do máximo ou mínimo de uma função de uma ou mais variáveis, as quais podem estar sujeitas a certas restrições.

O emprego das técnicas de Programação Matemática apresenta as seguintes vantagens [TABAK, 71]:

- . Capacidade de manipular eficientemente os vínculos de desigualdade no controle e nas variáveis de estado, onde outros métodos, como Cálculo de Variações e Princípio do Máximo, poderiam

levar a um problema com condições de contorno em dois pontos extremos (*Two-Point Boundary-Value Problem - TPBVP*), de difícil solução;

. Não requerem tanta memória de computador, como as técnicas de Programação Dinâmica.

Apesar dessas vantagens, podem existir problemas particulares, para os quais os outros métodos sejam mais adequados. Entretanto, para uma grande classe de problemas de controle ótimo, a Programação Matemática é mais eficiente, sendo que em alguns casos ela fornece a única solução possível [TABAK, 71].

O problema de Programação Matemática é o de encontrar o valor de n variáveis, x_1, \dots, x_n , de forma a satisfazer as equações (2.5-1) e (2.5-2), bem como minimizar ou maximizar a equação (2.5.3):

$$g_i(x_1, \dots, x_n) \geq 0, \quad i = 1, \dots, p \quad (2.5-1)$$

$$h_j(x_1, \dots, x_n) = 0, \quad j = 1, \dots, q \quad (2.5-2)$$

$$\phi(x_1, \dots, x_n) = J \quad (2.5-3)$$

As equações (2.5-1) e (2.5-2) são chamadas *restrições* e f é chamada *função objetivo*. Um ponto \mathbf{x}^* é dito ótimo, se ele satisfaz (2.5-1) a (2.5-3).

2.5.1. Ramificações da Programação Matemática

Dependendo da forma da função objetivo e das restrições opostas às variáveis, os problemas de Programação Matemática podem ser subdivididos em vários tipos, como os apresentados a seguir, a saber: Programação Linear, Quadrática, Geométrica, Não Linear, Inteira, Estocástica e Dinâmica.

1- Programação Linear

O problema será chamado de Programação Linear se $g(\mathbf{x})$, $h(\mathbf{x})$ e $f(\mathbf{x})$, dados pelas equações (2.5-1) a (2.5-2), forem funções lineares da variável \mathbf{x} [LUEMBERGER, 73][PUCCINI, 80][SAKAROVITCH, 70]. Os problemas de Programação Linear podem ser eficientemente resolvidos pelo método simplex, ou simplex revisado [LUEMBERGER, 73][PUCCINI, 80][SAKAROVITCH, 70]. Na solução de problemas de grande porte, que podem levar a matrizes com 10.000 linhas ou mais, as técnicas de partição e decomposição podem ser empregadas [LASHDON, 70][WHITE, 73].

2- Programação Quadrática

Se $g(\mathbf{x})$ e $h(\mathbf{x})$ são funções lineares e a função objetivo é uma função quadrática de \mathbf{x} , o problema é dito de Programação Quadrática [WHITE, 66]. Em Estatística, problemas de mínimos quadrados sujeitos a restrições, freqüentemente levam à Programação Quadrática. Os métodos para a solução de tais problemas são basicamente variações do método simplex.

3- Programação Geométrica

Na programação Geométrica [TABAK, 71], a função objetivo tem a forma:

$$\phi(\mathbf{x}) = \sum_{i=1}^P C_i \prod_{j=1}^n (x_j)^{a_{ij}} \quad (2.5.1-1)$$

sujeita às restrições:

$$x_j > 0 \quad j = 1, 2, \dots, n \quad (2.5.1-2)$$

onde

C_i = coeficientes constantes e positivos;

a_{ij} = constantes reais e

x_j = variáveis do problema.

Nesse caso, a solução do problema pode ser obtida pela solução de um conjunto de equações lineares [TABAK, 71].

4- Programação Não Linear

Um tipo de programação mais geral, que engloba os casos anteriores, é a Programação Não Linear [LASDON, 70] [TABAK, 71] [WHITE, 73]. O problema é considerado de Programação Não Linear caso ocorra pelo menos uma não linearidade nas restrições ou na função objetivo. Nesse caso, as funções **f**, **g** e **h** definidas em (2.5-1) a (2.5-2) são as mais gerais possíveis. As técnicas de Programação Não Linear podem, então, ser empregadas nos casos de Programação Linear, Geométrica e Quadrática. Entretanto, os algo-

ritmos específicos para cada caso, são mais eficientes que os de Programação Não Linear, quando aplicados a estes casos.

A solução do problema de controle ótimo freqüentemente exige a solução de um problema de otimização, em que tanto o funcional de custo quanto as restrições são não lineares. Nesses casos, os métodos de Programação Não Linear podem ser aplicados com sucesso [ABADIE, 78][AOKI, 71][BERTSEKAS, 74][FONG, 71][HASDORFF, 76][LASDON 67a][LASDON 67b][TABAK, 71][NOTON, 72][LASDON, 70][TRIPATHI, 70]. Por esse motivo, na seção 2.6, serão abordados os principais métodos e algoritmos de Programação Não Linear.

5- Programação Inteira

A Programação Inteira é uma técnica especial para a resolução de problemas de Programação Matemática, em que as variáveis podem assumir apenas valores inteiros [WHITE, 73][WISNER, 78][ZOUTENDIJK, 76]. Apesar da maioria dos trabalhos nesta área serem voltados para a Programação Linear Inteira, existem métodos para a solução dos casos não lineares. Um caso particular da Programação Inteira se apresenta na Programação Mista, em que parte das variáveis assumem valores inteiros e parte pode assumir valores reais quaisquer.

6- Programação Estocástica

Caso as variáveis tenham natureza aleatória, o problema é chamado de Programação Estocástica [TABAK, 71][WHITE, 73].

7- Programação Dinâmica

Alguns autores [ZOUTENDIJK, 76] [WHITE, 73] [WISNER, 78] consideram a Programação Dinâmica, da seção 2.4, como um problema especial de Programação Matemática.

2.6. Métodos de Otimização Irrestrita

Os casos de Programação Não Linear podem se apresentar sob duas formas distintas. A forma mais geral, chamada Programação Não Linear Restrita [HIMMELBLAU, 72] [WHITE, 72], trata do problema de otimização de equações do tipo (2.5-3), sujeitas a restrições impostas por (2.5-1) e (2.5-2). Na ausência destas restrições, tem-se o caso da Programação Não Linear Irrestrita.

Nos métodos de otimização irrestrita [FLETCHER, 69] [HIMMELBLAU, 72] [WHITE, 72] [POWELL, 70], busca-se o ótimo de uma função de n variáveis, função esta não sujeita a restrições. Nesse caso, busca-se um valor ótimo J^* , que pode ser um máximo ou um mínimo, para a equação (2.5-3) e não se tem equações do tipo (2.5-1) e (2.5-2).

No decorrer de todo o trabalho, serão tratados somente casos de minimização. A maximização pode ser obtida diretamente observando que:

$$\min(J) = \max(-J),$$

onde J é o funcional de custo que se pretende otimizar.

O método do gradiente conjugado é uma técnica de minimização irrestrita. Entretanto, uma generalização do método foi apresentada por Pagurek & Woodside [PAGUREK, 68] para tratar o caso com restrições na variável de controle $u(t)$ (saturação). Tal generalização consiste basicamente em transformar o problema com restrições em um irrestrito, que pode daí ser resolvido da maneira tradicional.

Os métodos de otimização irrestrita podem ser divididos em duas classes:

(a) métodos que se utilizam das derivadas da função objetivo [AOKI, 71] [HIMMELBLAU, 72] [FLETCHER, 69] [POWELL, 70];

(b) métodos que não usam as derivadas da função objetivo, chamados métodos diretos [AOKI, 71] [HIMMELBLAU, 72] [FLETCHER, 69] [POWELL, 64] [POWELL, 70] [ROSENBROCK, 70] [ZANGWILL, 67].

Os métodos que empregam o gradiente e/ou derivadas segundas, convergem mais rapidamente que aqueles diretos. Entretanto, em problemas com um grande número de variáveis, pode-se tornar difícil a obtenção do gradiente ou derivadas segundas, quer por meio de uma expressão algébrica ou por métodos numéricos. Em tais casos, melhores resultados podem ser obtidos pelos métodos diretos. Na prática, a escolha de um método vai depender do problema considerado, da velocidade de convergência, confiabilidade, precisão do método e experiência do usuário.

Dentre os métodos de otimização que se utilizam de derivadas da função objetivo, os métodos de gradiente são os mais

utilizados, por não necessitarem de derivadas segundas. Entretanto, eles apresentam algumas desvantagens, como convergência lenta e uma certa tendência a oscilação quando próximo do ponto ótimo. Tais desvantagens não ocorrem com o método do gradiente conjugado, que apresenta propriedades de convergência próximas aos métodos que usam derivadas segundas da função objetivo.

Alguns métodos de gradiente, inclusive o método do gradiente conjugado, que é a principal ferramenta para a solução dos problemas de controle ótimo tratados neste trabalho, serão apresentados nas subseções que seguem.

2.6.1. Método do Gradiente Ótimo

O método do gradiente é bastante simples e fornece uma maneira quase que intuitiva de se calcular o mínimo de um funcional $f(\mathbf{x})$. Considerando que o gradiente de um funcional, calculado em um ponto \mathbf{x}_0 , dá a direção de maior crescimento do funcional em \mathbf{x}_0 , o sentido contrário ao do gradiente resulta, portanto, na maior diminuição do valor do funcional.

Um algoritmo de minimização pelo método do gradiente é dado por:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \mathbf{g}(\mathbf{x}_k) \quad (2.6.1-1)$$

onde

$\mathbf{g}(\mathbf{x}_k)$ é o gradiente da função objetivo f em \mathbf{x}_k e

λ_k é um escalar que determina o passo a ser dado na iteração k , na direção determinada pelo gradiente.

Normalmente, o algoritmo definido por (2.6.1-1) nem sempre chega ao ponto de mínimo \mathbf{x}^* em um único passo, necessitando, nesses casos, de várias iterações do método. O escalar λ , que a cada iteração determina o comprimento do passo, pode ser obtido de várias maneiras, inclusive como uma constante unitária.

Um método é chamado de gradiente com passo ótimo ou, simplesmente, *gradiente ótimo*, se, a cada iteração, λ satisfizer:

$$\frac{df(\mathbf{x}_k - \lambda_k \mathbf{g}(\mathbf{x}_k))}{d\lambda} = 0 \quad (2.6.1-2)$$

Vários métodos analíticos podem ser empregados para a obtenção de λ de forma a satisfazer (2.6.1-2). O método empregado nos algoritmos deste trabalho será descrito no próximo capítulo.

O método do gradiente ótimo, segundo Himmelblau [HIMMELBLAU, 72], quando aplicado a uma função objetivo com diferenciabilidade pelo menos de terceira ordem, converge no limite quando $k \rightarrow \infty$. Na prática, não é necessário ter um número infinito de iterações para se chegar a uma razoável aproximação do mínimo. Esta aproximação rápida pode não ocorrer se a função objetivo tiver, próximo ao mínimo, a forma de um vale estreito e alongado. Nesse caso, o método vai apresentar um comportamento oscilatório, resultando em uma convergência muito lenta, inviabilizando sua aplicação em casos práticos.

2.6.2. Método das Direções Conjugadas

Os métodos que empregam derivadas parciais de ordem maior que o gradiente levam em consideração informações a respeito da curvatura da função objetivo [FLETCHER, 64]. Esses métodos não apresentam os problemas de convergência lenta e comportamento oscilatório, típicos do método do gradiente ótimo. Considera-se que os mesmos tenham convergência quadrática, significando que, desprezados os erros de arredondamento, o mínimo de uma função de n variáveis é localizado em um número finito de iterações - se a função objetivo for quadrática, geralmente em n passos.

O método das direções conjugadas [FLETCHER, 64] [LUEMBERGER, 73] [HASDORFF, 76] [HIMMELBLAU, 72], apresentado a seguir, possui as propriedades mencionadas.

Considere uma função objetivo $\phi(\mathbf{x})$, que possa ser aproximada por uma função $F(\mathbf{x})$, como resultado de uma expansão por séries de Taylor, em torno do ponto de mínimo \mathbf{h} , com termos de segunda ordem. Assim,

$$\phi(\mathbf{x}) \approx F(\mathbf{x}) = F(\mathbf{h}) + \frac{1}{2} \langle (\mathbf{x} - \mathbf{h}), A(\mathbf{x} - \mathbf{h}) \rangle \quad (2.6.2-1)$$

onde A é o operador derivada segunda calculado no ponto de mínimo.

Se o operador linear A é simétrico e definido positivo, então $F(\mathbf{x})$ possui um único mínimo [FLETCHER, 64] [HASDORFF, 76].

Esse mínimo é determinado pelo método das direções conjugadas da seguinte maneira: a partir de uma estimativa inicial \mathbf{x}_0 do ponto de mínimo, constrói-se uma seqüência,

$$\{\mathbf{x}_i\} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_i, \dots] \quad (2.6.2-2)$$

onde

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i \quad i = 0, 1, 2, \dots, n-1 \quad (2.6.2-3)$$

Na expressão acima, α_i é escolhido por meio de uma busca unidimensional, de forma a minimizar $F(\mathbf{x}_i + \alpha \mathbf{p}_i)$ a cada iteração. Desta forma, α_i satisfaz:

$$\left. \frac{df}{d\alpha} (\mathbf{x}_i + \alpha \mathbf{p}_i) \right|_{\alpha=\alpha_i} = F'(\mathbf{x}_i + \alpha_i \mathbf{p}_i) \cdot \mathbf{p}_i = 0 \quad (2.6.2-4)$$

Se $\mathbf{g}(\mathbf{x}_{i+1})$ é o gradiente da função $F(\mathbf{x})$ no ponto \mathbf{x}_{i+1} , segue da definição de gradiente de um funcional [HASDORFF, 76] e da equação (2.6.2-4) que:

$$\langle \mathbf{g}(\mathbf{x}_{i+1}), \mathbf{p}_i \rangle = 0 \quad (2.6.2-5)$$

As direções de busca \mathbf{p}_i são direções "A conjugadas", onde "A conjugada" significa que os vetores \mathbf{p}_i são conjugados com relação ao operador linear A, ou seja:

$$\begin{aligned} \langle \mathbf{p}_i, A\mathbf{p}_i \rangle &= 0 & \text{se } \mathbf{p}_i &= \mathbf{p}_j \\ \langle \mathbf{p}_i, A\mathbf{p}_i \rangle &\neq 0 & \text{se } \mathbf{p}_i &\neq \mathbf{p}_j \end{aligned} \quad (2.6.2-6)$$

As referências [FLETCHER, 64][LUEMBERGER, 73][HASDORFF, 76] e [HIMMELBLAU, 72] mostram que o método descrito pela equação (2.6.2-3), satisfazendo (2.6.2-4) e (2.6.2-6), fornece o mínimo de uma função quadrática em pelo menos n passos.

A convergência quadrática é importante, visto que, uma função genérica qualquer, próximo ao ponto de mínimo, pode ser aproximada por uma função quadrática, através de séries de Taylor. A convergência será garantida, embora executando um número maior que n de iterações.

A geração do conjunto de direções "A conjugadas" torna-se problemática quando a função $F(\mathbf{x})$ e o gradiente $\mathbf{g}(\mathbf{x})$ são definidos numericamente, de forma que o operador linear A não pode ser obtido explicitamente. Para resolver esse problema, foram desenvolvidos métodos como o de Davidon-Fletcher-Powell (DFP), o das tangentes paralelas [AOKI, 71][HIMMELBLAU, 72] e o método do gradiente conjugado [HASDORFF, 76][FLETCHER, 64]. Todos esses são métodos de direções conjugadas, diferindo apenas no modo como as direções são calculadas.

Na referência [BOX, 66], foram comparados oito métodos diferentes para a minimização de funções com duas, três, cinco, dez e vinte variáveis. Foram analisados tantos os métodos diretos como os indiretos, inclusive os métodos indiretos do gradiente conjugado e o de DFP. Dentre todos, o método de DFP mostrou-se mais eficiente para a minimização de funções.

O método DFP [FLETCHER, 63] é uma reformulação do método original de Davidon [DAVIDON, 59], proposto por Fletcher e Powell. Nesse método, as direções A conjugadas \mathbf{p}_i são dadas por:

$$\mathbf{p}_i = -\mathbf{H}_i \mathbf{g}_i \quad (2.6.2-7)$$

onde \mathbf{H}_i para $i = 1, 2, 3, \dots$ é uma sequência de matrizes simétricas positivas definidas, geradas por:

$$\mathbf{H}_{i+1} = \mathbf{H}_i + \frac{\mathbf{p}_i \mathbf{p}_i^T}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} - \frac{\mathbf{H}_i \mathbf{y}_i \mathbf{y}_i^T \mathbf{H}_i}{\mathbf{y}_i^T \mathbf{H}_i \mathbf{y}_i} \quad (2.6.2-8)$$

com

$$\mathbf{y}_i = \mathbf{g}_{i+1} - \mathbf{g}_i \quad (2.6.2-9)$$

e \mathbf{H}_0 é uma matriz arbitrária positiva definida (usualmente, a matriz identidade).

A aplicação deste método requer a armazenagem da matriz \mathbf{H} , o que, computacionalmente, pode representar um inconveniente.

niente, podendo mesmo tornar proibitivo o seu uso, já que, dependendo do número de variáveis, há a necessidade de se reservar um grande espaço na memória para a armazenagem de H, aumentando também o tempo de processamento.

Assim, na utilização do método DFP em um problema em que o gradiente tenha (n) componentes, a matriz H vai requerer a armazenagem de (n x n) componentes. Por exemplo, no caso de funções contínuas, utilizando-se 200 componentes no método do gradiente; são necessárias 40.000 posições de memória para armazenar os elementos de H.

Diante desse fato, é desejável a existência de um método que, mesmo não tendo a eficiência do método DFP, ainda mantenha as características dos métodos de direções conjugadas. O método do gradiente conjugado da próxima seção possui tais características.

2.6.3. O Método do Gradiente Conjugado

O Método do Gradiente Conjugado foi desenvolvido originalmente por Hestenes e Stiefel [HESTENES, 52], para a solução em n passos de um conjunto de equações lineares simultâneas do tipo:

$$A\mathbf{x} = \mathbf{b} \quad (2.6.3-1)$$

sendo A uma matriz de ordem $n \times n$ de coeficientes, simétrica e positiva definida, e \mathbf{x} um vetor de ordem $n \times 1$.

A solução de tal conjunto de equações fornece um vetor \mathbf{x} tal que:

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mathbf{h}) = \mathbf{0} \quad (2.6.3-2)$$

com

$$\mathbf{b} = \mathbf{A}\mathbf{h} \quad (2.6.3-3)$$

A equação (2.6.3-2) é exatamente a condição para que \mathbf{x} seja o mínimo de (2.6.2-1).

O algoritmo, em sua forma geral [LUEMBERGER, 73][HAS-DORFF, 76][AOKI, 71][HIMMELBLAU, 72][FLETCHER, 64], pode ser descrito pelo seguinte conjunto de equações:

$$\mathbf{p}_0 = -\mathbf{g}(\mathbf{x}_0) = -\mathbf{g}_0 \quad (2.6.3-4)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i \quad (2.6.3-5)$$

onde $(\alpha_i > 0)$ minimiza

$$F(\mathbf{x}_i + \alpha_i \mathbf{p}_i) \quad (2.6.3-6)$$

e

$$\mathbf{p}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{p}_i \quad (2.6.3-7)$$

com

$$\beta_i = \frac{\langle \mathbf{g}_{i+1}, \mathbf{g}_{i+1} \rangle}{\langle \mathbf{g}_i, \mathbf{g}_i \rangle} \quad (2.6.3-8)$$

Na equação (2.6.3-4), \mathbf{x}_0 é uma estimativa inicial arbitrária do mínimo.

Esse algoritmo caracteriza um método de direções conjugadas e apresenta convergência quadrática [HASDORFF, 76] [RALSTON, 76], significando que uma função quadrática de ordem n vai convergir em n passos.

Tal característica é desejável, uma vez que, próximo do mínimo, uma função genérica qualquer pode ser aproximada por uma função quadrática, garantindo a convergência do método, mesmo quando aplicado a funções genéricas.

Pode ocorrer um problema com a razão de convergência do algoritmo do gradiente conjugado, quando ele é aplicado em funções que apresentam curvas de níveis semelhantes à chamada "função banana" de Rosenbrock [ROSENBROCK, 70] [HASDORFF, 76], mostrada na figura 2.1, cujo nome se deve à forma de suas curvas de níveis lembrarem uma banana, como mostra a figura 2.2. Devido à forma particular das curvas de níveis, as direções "A conjugadas" \mathbf{p}_i são aproximadamente paralelas, resultando em pontos \mathbf{x}_i muito próximos e, conseqüentemente, em uma convergência muito lenta.

Os métodos que fazem uso de derivada segunda da função objetivo levam em conta a curvatura da função próximo ao mínimo, levando a uma maior velocidade de convergência. Um desses métodos, proposto por Pagurek [PAGUREK, 68], assemelha-se ao método de Davidon-Fletcher-Powell. Já o método proposto por Hasdorff [HASDORFF, 76], transforma o espaço do funcional de custo, de forma a

deixar as curvas de níveis do funcional aproximadamente esféricas, facilitando a convergência - este método é chamado de gradiente conjugado escalonado.

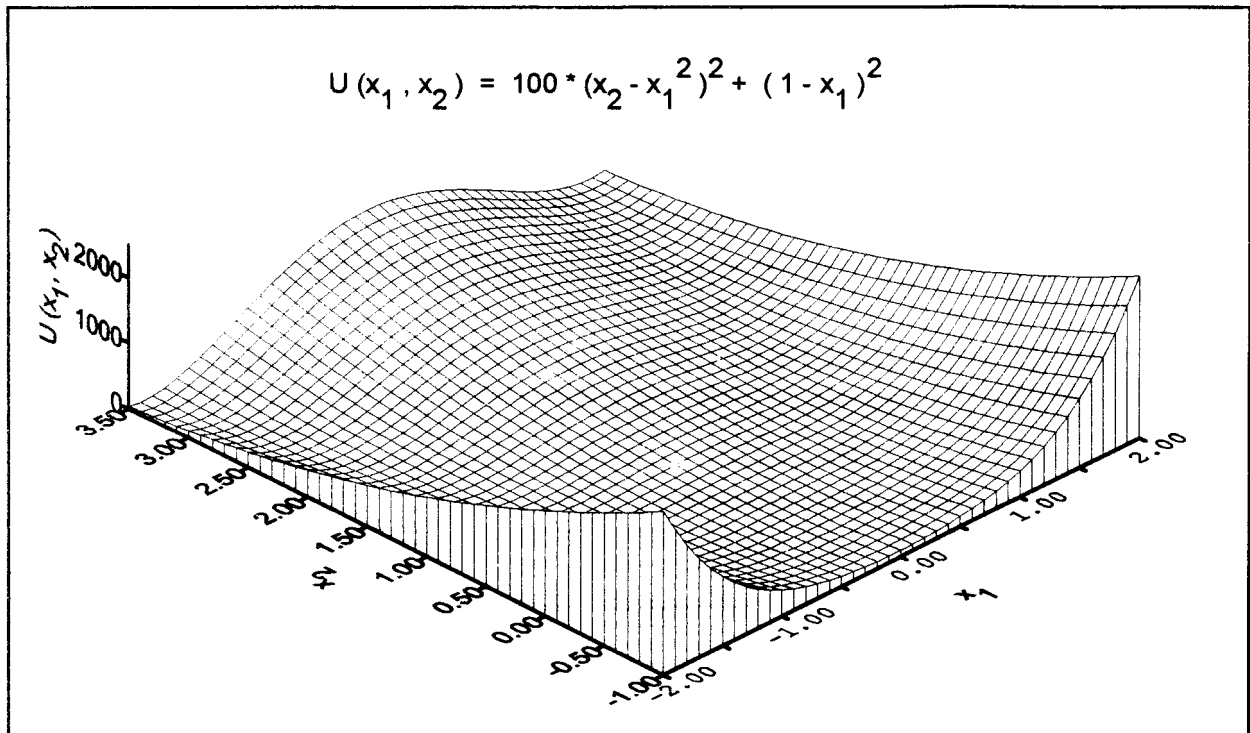


Figura 2.1 - "Função Banana" de Rosenbrock

Todavia, tais métodos apresentam as mesmas desvantagens do método de Davidon-Fletcher-Powell: requerem muita memória do computador. No método de Pagurek o operador derivada segunda da função objetivo deve ser armazenado e no método do gradiente conjugado escalonado, a matriz de transformação do subespaço original para o subespaço escalonado também deve ser armazenada.

Uma solução mais simples foi proposta por Fletcher e Reeves [FLETCHER, 64], para tratar com funções que apresentam características semelhantes às da função de Rosenbrock. Nesse

processo é utilizado o método do gradiente conjugado, mas, periodicamente, a direção de busca \mathbf{p}_i é revertida para a do gradiente ótimo $-\mathbf{g}_i$. A reversão é implementada reiniciando-se o processo a partir do valor corrente \mathbf{x}_i , descartando-se a informação prévia transmitida pela direção \mathbf{p}_i . Desde que as reinicializações não sejam mais freqüentes que

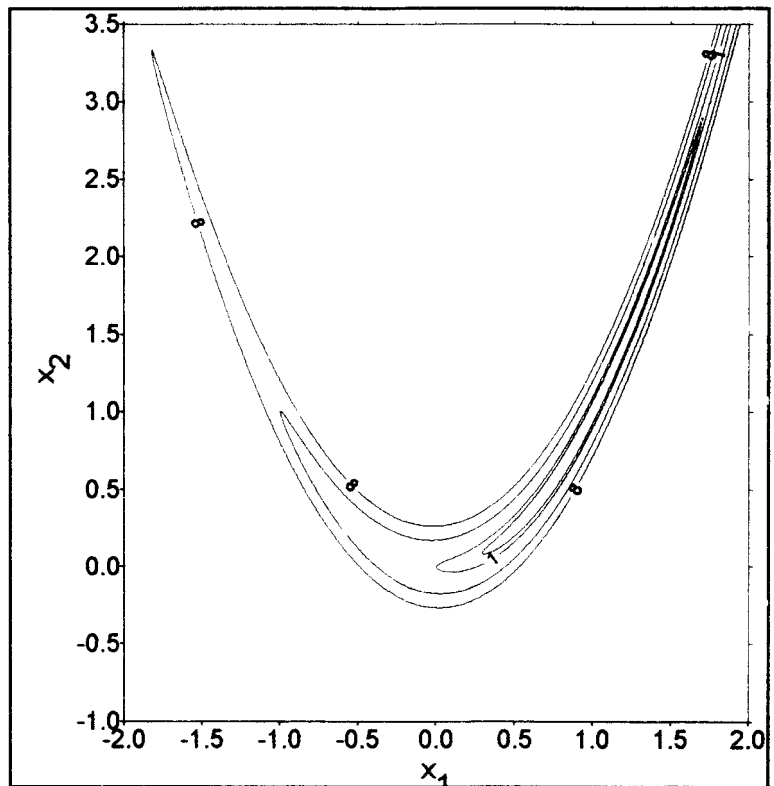


Figura 2.2 - Curvas de Níveis da "Função Banana"

as \underline{n} iterações, o método mantém as propriedades de convergência quadrática. A sugestão de Fletcher e Reeves é no sentido de se reinicializar o processo a cada $\underline{n+1}$ iterações.

Considerando que esse método requer menos memória do computador, a velocidade de execução será maior. No próximo capítulo, seu desempenho será comparado ao método do gradiente conjugado normal.

CAPÍTULO 3

Generalização do Método do Gradiente Conjugado para Solução de Problemas de Controle Ótimo

3.1. Introdução

A finalidade deste capítulo é apresentar uma generalização do método do gradiente conjugado, descrito no capítulo anterior, para que ele possa ser usado como ferramenta na solução de problemas de controle ótimo, permitindo fazer, em sistemas não lineares, o que antes só era possível nos sistemas lineares.

O interesse na generalização do método do gradiente conjugado se deve à sua simplicidade, clareza e abrangência, podendo ser aplicado a uma classe de problemas de otimização que transcendem a área de controle. Essas características são ressaltadas mesmo quando comparadas às de outros métodos numéricos

empregados na solução por computador de problemas de controle ótimo.

Vários autores propuseram modos de generalização do método do gradiente conjugado, visando sua extensão à solução dos problemas de controle ótimo. Entre esses autores encontram-se Lasdon, Mitter e Waren [LASDON, 67b], Pagurek e Woodside [PAGUREK, 68], Noton [NOTON, 72], Hasdorff [HASDORFF, 76] e, mais recentemente, Treiman [TREIMAN, 90].

Este capítulo é dividido basicamente em duas partes. Na primeira, seção 3.2, é apresentada uma forma de generalização do método do gradiente conjugado, seguindo a abordagem enfocada por Hasdorf. Na segunda parte, seção 3.3, o caso do controle ótimo de um motor de corrente contínua (DC) é analisado, a fim de servir de exemplo e padrão para uma análise de sensibilidade dos diversos parâmetros.

3.2. Generalização do Método do Gradiente Conjugado para o Espaço de Hilbert

O desenvolvimento, apresentado no capítulo anterior, para o método do gradiente conjugado aplica-se à minimização de funções com domínio \mathbb{R}^n , tratando-se, nesse caso, de otimização com dimensão finita [NOTON, 72]. Entretanto, para emprego do método em problemas de controle ótimo, faz-se necessária a apli-

cação em espaços mais gerais, envolvendo o problema de otimização com dimensão infinita. Tal generalização consiste em estender o método para o espaço de Hilbert (mais geral que o \mathbb{R}^n - vide *APÊNDICE A*), de forma que as entradas de controle possam ser representadas por funções contínuas, conjuntos de parâmetros, entradas amostradas, etc. [HASDORFF, 76][LUEMBERGER, 73].

A generalização do método envolve, basicamente, a identificação de uma expressão para o gradiente do funcional de custo no espaço em questão.

3.2.1. Expressão para o Gradiente de um Funcional

Seja F um funcional do espaço de Hilbert \mathcal{H} em \mathbb{R}^1 , diferenciável e \mathbf{x}_0 , \mathbf{t} , e \mathbf{z} da forma:

$$\mathbf{t} = \epsilon \mathbf{z} \quad \text{onde } \epsilon = \text{parâmetro escalar} \quad (3.2.1-1)$$

$$\|\mathbf{z}\| = 1 \quad (3.2.1-2)$$

Fazendo a expansão do funcional F em série de Taylor, em torno do ponto \mathbf{x}_0 , tem-se:

$$F(\mathbf{x}_0 + \epsilon \mathbf{z}) = F(\mathbf{x}_0) + \epsilon \langle \mathbf{g}(\mathbf{x}_0), \mathbf{z} \rangle + 1/2 \epsilon^2 \langle \mathbf{z}, \mathbf{A}(\mathbf{x}_0) \mathbf{z} \rangle + O^2(\epsilon) \quad (3.2.1-3)$$

onde

$\mathbf{g}(\mathbf{x}_0) \in \mathcal{H}$ é o gradiente de F em \mathbf{x}_0 ,
 $A(\mathbf{x}_0)$ é o operador derivada segunda e
 $O^2(\mathbf{t})$ são os termos de ordem superiores.

Diferenciando ambos os membros de (3.2.1-3) em relação a ϵ e calculando em $\epsilon = 0$, obtém-se:

$$\left. \frac{d}{d\epsilon} F(\mathbf{x}_0 + \epsilon \mathbf{z}) \right|_{\epsilon=0} = \langle \mathbf{g}(\mathbf{x}_0), \mathbf{z} \rangle \quad (3.2.1-4)$$

A equação acima fornece o método básico para a obtenção do gradiente, o qual consiste em realizar as operações definidas do lado esquerdo de (3.2.1-4) sobre o funcional F e identificar $\mathbf{g}(\mathbf{x}_0)$ usando a definição de produto escalar no espaço em que se deseja obter o controle ótimo.

A equação (3.2.1-4) é uma equação fundamental no cálculo de variações, onde ela é usada para se obter as condições necessárias para o mínimo, além de desempenhar igual papel na teoria de otimização. Na literatura ela é chamada de diferencial de Gateaux [HASDORFF, 76].

3.2.2. O Gradiente do Funcional de Custo para Sistemas de Controle

O sistema, para o qual o gradiente será calculado, é descrito pela seguinte equação dinâmica:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) ; \quad \mathbf{x}(t_0) = \mathbf{c} \quad (3.2.2-1)$$

com $(\mathbf{x} \in \mathbb{R}^n)$ e $(\mathbf{u} \in U)$, onde

U é um espaço de Hilbert, domínio do funcional de custo;

$\mathbf{x}(t_0)$ é o estado inicial do sistema;

\mathbf{c} é uma constante e

\mathbf{u} é a entrada de controle do sistema.

Da equação (3.2.2-1), \mathbf{f} é uma aplicação com domínio em $\mathbb{R}^n \times U$ e codomínio em \mathbb{R}^n . Além disso, \mathbf{f} deve ter derivadas parciais segundas contínuas com relação a \mathbf{x} e \mathbf{u} e, para um dado \mathbf{u} , a equação (3.2.2-1) deve ter solução única no intervalo $[t_0, t_f]$.

O funcional de custo neste caso é:

$$J = J(\mathbf{u}) = \phi(\mathbf{x}(t_f, \mathbf{u})) \quad (3.2.2-2)$$

O funcional de custo é uma função composta, pois dada uma entrada de controle $\mathbf{u} \in U$ e $\mathbf{x}(t_f, \mathbf{u}) \in \mathbb{R}^n$, que é obtido resolvendo a equação (3.2.2-1), o funcional $J[\mathbf{u}] \in \mathbb{R}^1$ é calculado por meio de $\phi(\mathbf{x}(t_f, \mathbf{u}))$, como pode ser visto na figura 3.1.

O problema do controle ótimo é encontrar um $\mathbf{u}^* \in U$, que minimize J .

É importante observar que não há restrições sobre as variáveis de estado ou de controle. Caso houvesse, uma técnica

especial deveria ser empregada para transformar o problema em um irrestrito [PAGUREK, 68], para que o método do gradiente conjugado pudesse ser empregado.

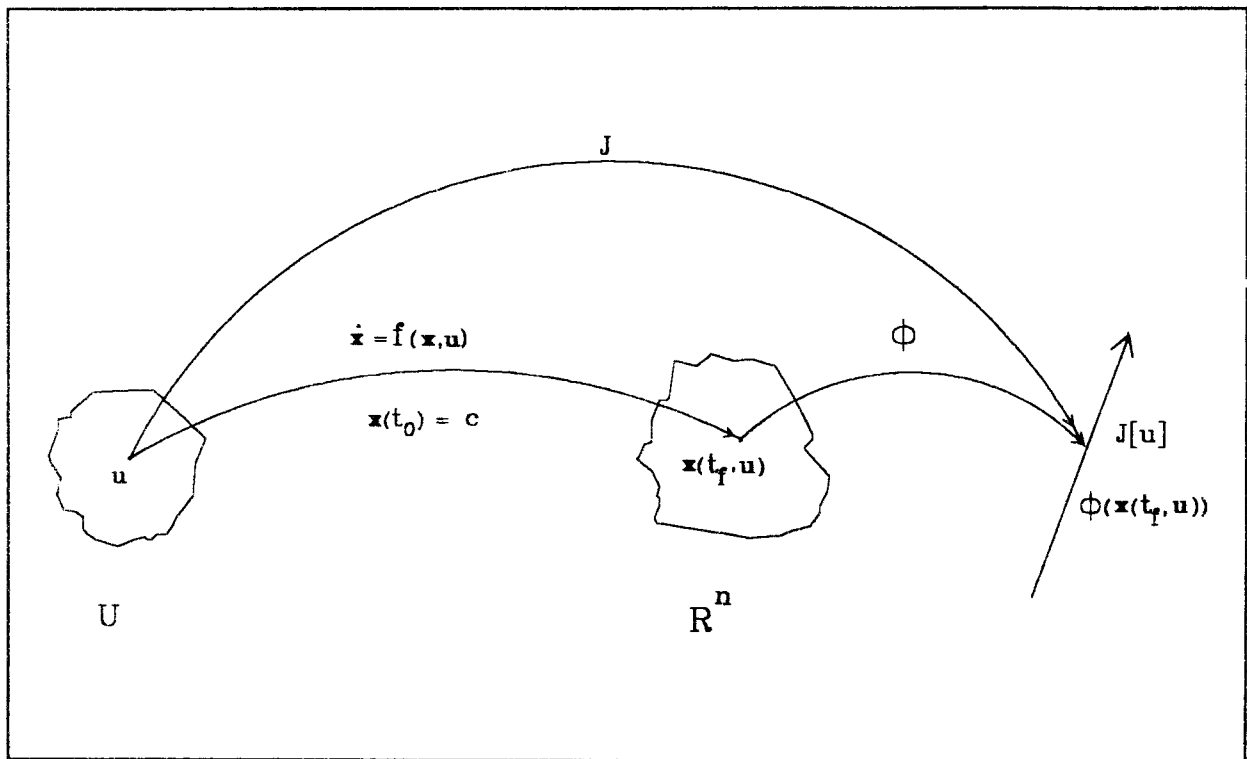


Figura 3.1 - O Funcional de Custo representado como um operador composto

Para o cálculo do funcional $J[\mathbf{u}]$ é necessário calcular o gradiente $\mathbf{g}(\mathbf{u})$, que é obtido usando $J[\mathbf{u}]$, da equação (3.2.2-2), na equação (3.2.1-4), resultando:

$$\begin{aligned} \frac{d}{d\epsilon} [J[\mathbf{u} + \epsilon \mathbf{z}]] \big|_{\epsilon=0} &= \frac{d}{d\epsilon} [\phi(\mathbf{x}(t_f, \mathbf{u} + \epsilon \mathbf{z}))] \big|_{\epsilon=0} = & (3.2.2-3) \\ &= \left\langle \nabla_{\mathbf{x}} \phi(\mathbf{x}(t_f, \mathbf{u} + \epsilon \mathbf{z})), \frac{d}{d\epsilon} [\mathbf{x}(t_f, \mathbf{u} + \epsilon \mathbf{z})] \right\rangle \big|_{\epsilon=0} \end{aligned}$$

Comparando os dois lados da equação (3.2.1-4) e usando:

$$\frac{d}{d\epsilon} [\mathbf{x}(t_f, \mathbf{u} + \epsilon \mathbf{z})] |_{\epsilon=0} \equiv \frac{d}{d\epsilon} [\mathbf{x}(t_f, \mathbf{u})] \quad (3.2.2-4)$$

obtém-se:

$$\frac{d}{d\epsilon} [J[\mathbf{u} + \epsilon \mathbf{z}]] |_{\epsilon=0} = \langle \mathbf{g}(\mathbf{u}), \mathbf{z} \rangle = \left\langle \nabla_{\mathbf{x}} \phi(\mathbf{x}(t_f, \mathbf{u})), \frac{d}{d\epsilon} [\mathbf{x}(t_f, \mathbf{u})] \right\rangle \quad (3.2.2-5)$$

Esta igualdade permite a identificação do gradiente $\mathbf{g}(\mathbf{u})$, que para cada espaço de controle considerado, fica reduzido ao cálculo da equação (3.2.2-4).

No problema do controle ótimo, alguns espaços de Hilbert típicos para as entradas de controle \mathbf{u} são: Funções Contínuas, Conjuntos de Parâmetros, Entradas Amostradas, Modulação por Largura de Pulso e espaço de Condições Iniciais. Na referência [FISCHER, 83], as expressões para o gradiente nos quatro primeiros espaços, foram deduzidas de forma detalhada. Visto que o caso de funções contínuas foi usado como parâmetro para a medida do desempenho dos vários processadores, este é o caso que será tratado a seguir.

3.3. O Problema de Teste

O objetivo deste trabalho é a análise de desempenho de processadores, para aplicação no controle ótimo de processos em tempo real. Nesse caso, um dos problemas fundamentais é a velocidade de processamento. Essa velocidade pode ser medida de várias maneiras, como, por exemplo, usando técnicas de "Benchmark" específicas para a análise de operações matemáticas inteiras e em ponto flutuante, bem como medindo a velocidade de acesso à memória, ao disco e ao barramento.

Outra forma de se medir a velocidade de processamento é através da medida direta do desempenho dos algoritmos específicos da aplicação desejada. Essa é a forma escolhida para esta análise, onde o algoritmo usado é o do método do gradiente conjugado.

Para efeitos de testes, foi escolhido o problema da obtenção do controle ótimo de um motor de corrente contínua (DC), controlado pelo campo. Nesse caso a tensão de campo é a entrada de controle, que é modelada como uma função contínua do tempo $u(t)$.

3.3.1. O Gradiente no Espaço de Funções Contínuas

Uma entrada de controle neste espaço é contínua e tem a forma:

$$u = u(t) , \quad t \in [t_0, t_f] \quad (3.3.1-1)$$

Para obter o gradiente do funcional de custo, é usada

a equação (3.2.2-5), onde o termo $(dx/d\epsilon \text{ de } t_f)$ deve ser calculado.

Integrando o sistema de equações diferenciais (3.2.2-1), obtém-se:

$$\mathbf{x}(t_f, u) = \mathbf{x}(t_0) + \int_{t_0}^{t_f} f(\mathbf{x}, u) d\tau \quad (3.3.1-2)$$

Usando a equação (3.2.2-4) e derivando (3.3.1-2) com relação a ϵ :

$$\frac{d}{d\epsilon} [\mathbf{x}(t_f, u + \epsilon z)] \big|_{\epsilon=0} = \quad (3.3.1-3)$$

$$= \frac{d}{d\epsilon} [\mathbf{x}(t_0)] \big|_{\epsilon=0} + \int_{t_0}^{t_f} \left[f_x(\mathbf{x}, u) \frac{d}{d\epsilon} [\mathbf{x}(t, u)] + f_u(\mathbf{x}, u) z(t) \right] dt \big|_{\epsilon=0}$$

onde $z = z(t)$, $t \in [t_0, t_f]$ e:

$$(f_x(\mathbf{x}, u))_{ij} = \frac{\partial f_i(\mathbf{x}, u)}{\partial x_j} \quad i, j = 1, 2, 3, \dots, n \quad (3.3.1-4)$$

$$(f_u(\mathbf{x}, u))_i = \frac{\partial f_i(\mathbf{x}, u)}{\partial u} \quad i = 1, 2, 3, \dots, n \quad (3.3.1-5)$$

Eliminando-se alguns argumentos de (3.3.1-3) e voltando a usar a relação (3.2.2-4), segue a expressão:

$$\frac{d\mathbf{x}}{d\epsilon}(t_f, u) = \frac{d\mathbf{x}}{d\epsilon}(t_0) + \int_{t_0}^{t_f} f_x \frac{d\mathbf{x}}{d\epsilon}(t, u) + f_u z(t) dt \quad (3.3.1-6)$$

Para o cálculo do gradiente é necessário resolver a equação integral acima. Isto é feito considerando que a mesma deve ser válida para qualquer $t_f > t_0$. Fazendo $t_f = t$ e diferenciando com relação a t , obtém-se:

$$\frac{d}{dt} \frac{d\mathbf{x}}{d\epsilon}(t, u) = f_x \frac{d\mathbf{x}}{d\epsilon}(t, u) + f_u z(t) \quad (3.3.1-7)$$

A equação acima é uma equação diferencial não homogênea de primeira ordem, do tipo:

$$\dot{\mathbf{x}} = A(t)\mathbf{x} + B(t)u \quad (3.3.1-8)$$

onde:

$$\mathbf{x} = \frac{d\mathbf{x}}{d\epsilon}(t, u) \quad (3.3.1-9)$$

$$A(t) = f_x \quad (3.3.1-10)$$

$$B(t) = f_u \quad (3.3.1-11)$$

$$u = z(t) \quad (3.3.1-12)$$

A solução de (3.3.1-8) é obtida resolvendo-se a equação homogênea (a equação sem a parcela $\mathbf{B}(t)u$) e então introduzindo-se as variáveis de controle. Assim, obtém-se a seguinte expressão [CHEN, 70] [HASDORFF, 76] [NOTON, 72]:

$$\mathbf{X}(t) = \Phi(t, t_0) \mathbf{X}(t_0) + \int_{t_0}^t \Phi(t, \tau) \mathbf{B}(\tau) u(\tau) d\tau \quad (3.3.1-13)$$

onde $\Phi(t, t_0)$ é a matriz de transição [NOTON, 72], obtida da equação:

$$\frac{d\Phi}{dt}(t, t_0) = A(t) \Phi(t, t_0) \quad (3.3.1-14)$$

com

$$\Phi(t_0, t_0) = I \quad (3.3.1-15)$$

sendo I a matriz identidade ($n \times n$).

Substituindo as equações (3.3.1-9) a (3.3.1-12) em (3.3.1-13), tem-se:

$$\frac{d\mathbf{x}}{d\epsilon}(t, u) = \Phi(t, t_0) \frac{d\mathbf{x}}{d\epsilon}(t_0) + \int_{t_0}^t \Phi(t, \tau) \mathbf{f}_u \mathbf{z}(\tau) d\tau \quad (3.3.1-16)$$

com $\mathbf{x}(t_0) = \mathbf{c}$, segue de (3.2.2-5):

$$\langle g(u), \mathbf{z} \rangle = \left\langle \nabla_{\mathbf{x}} \Phi(\mathbf{x}(t_f, u)), \int_{t_0}^{t_f} \Phi(t_f, \tau) \mathbf{f}_u \mathbf{z}(\tau) d\tau \right\rangle \quad (3.3.1-17)$$

considerando $\langle \mathbf{x}, B\mathbf{y} \rangle = \langle B^T, \mathbf{y} \rangle$, com $B = \Phi(t_f, \tau) \mathbf{f}_u$, obtém-se:

$$\langle g(u), z \rangle = \int_{t_0}^{t_f} \langle \mathbf{f}_u^T \Phi^T(t_f, \tau) \nabla_x \Phi(\mathbf{x}(t_f, u)) z(\tau) \rangle d\tau \quad (3.3.1-18)$$

Definindo $\lambda(t)$ e notando que $z(\tau)$ é uma função escalar tem-se:

$$\lambda(t) \equiv \Phi^T(t_f, t) \nabla_x \Phi(\mathbf{x}(t_f, u)) \quad (3.3.1-19)$$

$$\langle g(u), z \rangle = \int_{t_0}^{t_f} \mathbf{f}_u^T \lambda(t) z(t) dt \quad (3.3.1-20)$$

Da definição do produto escalar no espaço de funções contínuas e da equação acima, segue que o gradiente do funcional de custo neste espaço é dado por:

$$g(u(t)) = \mathbf{f}_u^T \lambda(t); \quad t \in [t_0, t_f] \quad (3.3.1-21)$$

Para calcular o gradiente é necessário determinar $\lambda(t)$. Entretanto, ao invés de se usar a equação (3.3.1-19), é mais simples determinar $\lambda(t)$ através da solução da equação adjunta:

$$\dot{\lambda}(t) = -\mathbf{f}_x^T \lambda(t) \quad (3.3.1-22)$$

com

$$\lambda(t_f) = \nabla_x \phi(x(t_f, u)) \quad (3.3.1-23)$$

pois $\lambda(t)$ que resolve a equação (3.3.1-22) com a condição terminal (3.3.1-23) também resolve (3.3.1-19) [HASDORFF, 76].

3.3.2. Modelamento do Motor DC

O problema escolhido foi o controle ótimo da tensão de campo de um motor DC, de aproximadamente 5hp. Especificamente, o problema a ser resolvido é encontrar uma entrada de controle $u(t)$, que produza uma mudança em degrau de 10 radianos no eixo do motor e minimize um certo critério de desempenho. Neste caso, o controle ótimo não é calculado em tempo real.

O motor, com os principais parâmetros, estão ilustrados na figura 3.2, onde se tem:

- R,fH : resistência de campo;
- L,fH : indutância de campo;
- i,fH : corrente de campo;
- e,fH : tensão de campo;
- i,armH : corrente de armadura;
- B : coeficiente de amortecimento;
- J : momento de inércia;
- θ : posição do eixo (em radianos);
- K_t : constante de torque.

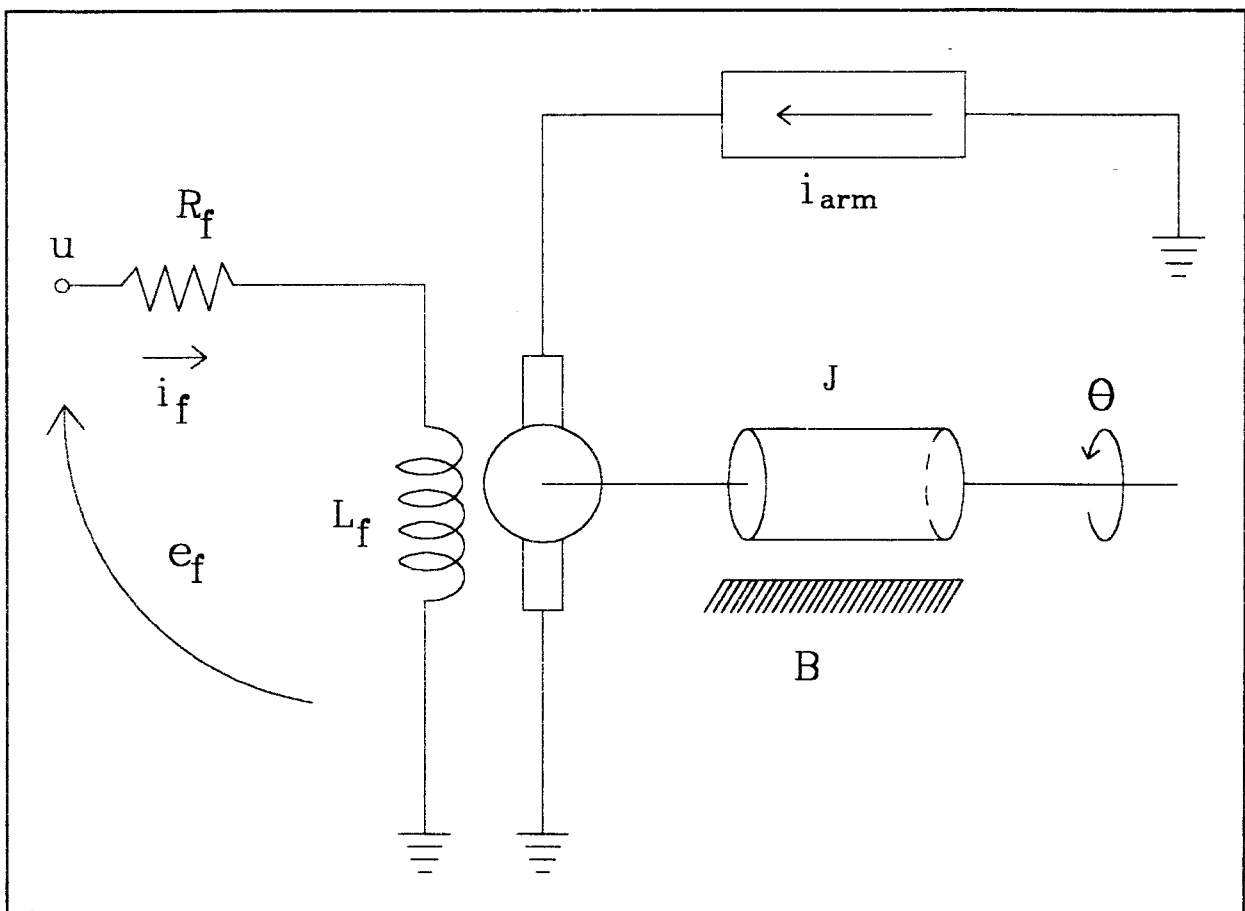


Figura 3.2 - Esquema e características do motor DC controlado pelo campo.

$R_f = 10 \, \Omega$; $L_f = 10 \, \text{h}$; $J = 2 \, \text{Kg.m}^2$; $B = 0,66 \, \text{Kg.m}^2/\text{seg}$; $K_t = 10 \, \text{N.m/A}$

Para o sistema considerado, as variáveis de estado são definidas por:

$$x_1 = \theta \quad (3.3.2-1)$$

$$x_2 = \dot{\theta} \quad (3.3.2-2)$$

$$x_3 = i_f \quad (3.3.2-3)$$

Com esta escolha de variáveis, o modelo do motor pode ser representado pelo seguinte conjunto de equações de estado:

$$\dot{x}_1 = x_2 \quad (3.3.2-4)$$

$$\dot{x}_2 = \frac{B}{J}x_2 + \frac{K_t}{J}x_3 \quad (3.3.2-5)$$

$$\dot{x}_3 = -\frac{R_f}{L_f}x_3 + \frac{u}{L_f} \quad (3.3.2-6)$$

Nessas expressões, u é a entrada de controle, representando a tensão de campo e_f .

Para que o eixo do motor se aproxime de uma variação em degrau de 10 rad, um critério de custo deve ser formulado de forma a ter um mínimo quando a saída do sistema (x_1) apresentar tal variação.

Desta forma, o funcional de custo é definido como a integral do quadrado do erro entre a posição real e a posição desejada. Nessa integral, é adicionado um termo de penalização do custo $Ru^2(t)$, para limitar a energia aplicada na entrada do motor. Essa limitação evita valores de tensão de entrada que não possam ser reproduzidos na prática. No funcional de custo são ainda acrescentados mais três termos de penalização, com ponderações W_1 , W_2 , W_3 , para garantir que o estado final seja atingido. Isso fornece um critério da forma:

$$J(u) = \int_{t_0}^{t_f} [(x_1 - 10)^2 + Ru^2] dt + [W_1 (x_1 - 10)^2 + W_2 x_2^2 + W_3 x_3^2] \big|_{t=t_f} \quad (3.3.2-7)$$

A equação integral acima pode ser resolvida definindo a nova variável de estado:

$$\dot{x}_4 = (x_1 - 10)^2 + Ru^2 \quad \text{com } x_4(0) = 0 \quad (3.3.2-8)$$

Substituindo (3.3.2-8) em (3.3.2-7), o critério de custo pode ser escrito como:

$$J(u) = \phi(\mathbf{x}(t_f)) = [x_4 + W_1 (x_1 - 10)^2 + W_2 x_2^2 + W_3 x_3^2] \big|_{t=t_f} \quad (3.3.2-9)$$

Substituindo os valores dos parâmetros do motor nas equações (3.3.2-4) a (3.3.2-6) e (3.3.2-8), tem-se os seguintes componentes para a equação de estado, considerando o estado inicial dado por $\mathbf{x}(0) = \mathbf{c} = 0$:

$$\dot{x}_1 = x_2 \quad (3.3.2-10)$$

$$\dot{x}_2 = -\frac{x_2}{3} + 5x_3 \quad (3.3.2-11)$$

$$\dot{x}_3 = -x_3 + 0,1u \quad (3.3.2-12)$$

$$\dot{x}_4 = (x_1 - 10)^2 + Ru^2 \quad (3.3.2-13)$$

Levando em consideração as equações de estado (3.3.2-10) a (3.3.2-13) e as equações (3.3.1-21) a (3.3.1-23), temos para o gradiente e para a equação adjunta:

$$g(u(t)) = 0,1\lambda_3(t) + 2Ru(t)\lambda_4(t) \quad t \in [t_0, t_f] \quad (3.3.2-14)$$

$$\lambda_1 = -2(x_1 - 10)\lambda_4 \quad (3.3.2-15)$$

$$\lambda_2 = -\lambda_1 + \frac{\lambda_2}{3} \quad (3.3.2-16)$$

$$\lambda_3 = -5\lambda_2 + \lambda_3 \quad (3.3.2-17)$$

$$\lambda_4 = 0 \quad (3.3.2-18)$$

com a condição terminal $\lambda(t_f)$ dada por:

$$\lambda(t_f) = [2w_1(x_1 - 10) \quad 2w_2x_2 \quad 2w_3x_3 \quad 1]^T \quad (3.3.2-19)$$

Obtidas as equações acima, foi desenvolvido um programa para a solução do problema do controle ótimo pelo método do gradiente conjugado para a motor DC. A implementação desse programa será descrita nas próximas seções.

3.4. O Procedimento Computacional

Utilizando os algoritmos do método do gradiente conjugado, descrito no capítulo anterior, e os procedimentos para a obtenção do gradiente no espaço de funções contínuas (sub-seção 3.3.1), foi implementado um programa para gerar as entradas de controle que otimizam o funcional de custo durante a operação do motor DC.

As entradas são obtidas em um intervalo de tempo determinado *a priori*, ou seja, o tempo final t_f é fixado. O espaço das entradas é o de funções contínuas, mas o procedimento pode ser estendido para outros espaços de entradas de controle [FISCHER, 83].

A dinâmica do sistema, para o qual as entradas de controle serão calculadas, é dada pela equação (3.3.2-10) a (3.3.2-13). O funcional de custo, para uma dada entrada de controle \mathbf{u} , é dado pela equação (3.3.2-9), sendo o gradiente obtido pela equação (3.3.2-14) e a equação adjunta pelas equações (3.3.2-15) a (3.3.2-19).

O programa localiza \mathbf{u}^* , tal que $J(\mathbf{u}^*)$ é um mínimo, iterativamente, segundo uma sequência $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_n$, onde \mathbf{u}_0 é uma estimativa inicial e $J(\mathbf{u}_{i+1}) < J(\mathbf{u}_i)$. O limite desta sequência é o valor \mathbf{u}^* procurado.

3.4.1. Implementação do Algoritmo do Gradiente Conjugado

A sequência $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_n$, onde $J(\mathbf{u}_{i+1}) < J(\mathbf{u}_i)$, é gerada basicamente pelo algoritmo do gradiente conjugado da subseção 2.6.3.

O método do gradiente conjugado requer o cálculo do gradiente, dado pela equação (3.3.2-14), que por sua vez, necessita da solução da equação adjunta (3.3.2-15) a (3.3.2-19). A solução da equação adjunta é obtida de t_f até t_0 , partindo da condição $\lambda(t_f)$ da equação (3.3.2-19). Essa condição terminal precisa da solução das equações de estado do sistema, (3.3.2-10) a (3.3.2-13), para obter $\mathbf{x}(t_f)$ a partir de $\mathbf{x}(t_0)$. Dessa forma, o procedimento para o cálculo do gradiente pode ser resumido pelo seguinte algoritmo:

- (1) Integrar as equações do sistema, (3.3.2-10) a (3.3.2-13), de t_0 a t_f , com $\mathbf{x}(t_0) = \mathbf{0}$, dada a entrada $u(t)$ para obter o estado do sistema $\mathbf{x}(t)$ neste intervalo;
- (2) Obtido $\mathbf{x}(t_f)$, integra-se as equações adjuntas dadas por (3.3.2-15) a (3.3.2-18), com a condição terminal (3.3.2-19), de t_f a t_0 , para obter $\lambda(t)$ neste intervalo;
- (3) Usando a equação (3.3.2-14), calcula-se o gradiente $g(u(t))$.

As equações diferenciais são resolvidas pelo algoritmo de *Runge-Kutta* de quarta ordem, usando o método de *Gill* [RALSTON, 76].

O algoritmo do gradiente conjugado pode ser desenvolvido nas seguintes etapas:

- (1) Partindo de uma estimativa \mathbf{u}_0 , calcula-se o gradiente $\mathbf{g}(J(\mathbf{u}_0)) = \mathbf{g}_0$ e toma-se um vetor \mathbf{p}_0 , tal que $\mathbf{p}_0 = -\mathbf{g}_0$ (equação (2.6.3-4));
- (2) Para $i = 0, 1, \dots, n$, executa-se os seguintes passos:
 - (2.1) Determina-se, por meio de uma busca unidimensional, o escalar $\alpha_i > 0$, que minimiza $J(\mathbf{u}_i + \alpha_i \mathbf{p}_i)$ (equação (2.6.3-6));
 - (2.2) Calcula-se $\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha_i \mathbf{p}_i$ (equação (2.6.3-5));
 - (2.3) Obtém-se o novo \mathbf{p}_{i+1} tal que $\mathbf{p}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{p}_i$ (equação (2.6.3-7)), com β_i dado por (2.6.3-8) e $\mathbf{g}_{i+1} = \mathbf{g}[J(\mathbf{u}_{i+1})]$.

A busca unidimensional do passo (2.1) destes procedimentos será detalhada na próxima seção.

A determinação do escalar β_i envolve o cálculo de dois

produtos escalares no espaço de funções contínuas. Nesse caso, tais produtos escalares são determinados pela integral de t_0 a t_f do produto das funções envolvidas. Essa integral é calculada pela regra de *Simpson* [McCracken, 64], cujo erro de integração é da ordem de h^4 , onde h é o intervalo de discretização da função. É importante observar que um erro de mesma ordem de grandeza ocorre no método de *Runge-Kutta*.

Nesse caso de funções contínuas, as entradas $u(t)$ e os gradientes $g_i(t)$ e $g_{i+1}(t)$ são armazenados na memória do computador, para cada ponto de discretização no intervalo $[t_0, t_f]$.

Todos os programas foram implementados na linguagem FORTRAN. Não há nenhuma restrição, entretanto, quanto à linguagem de programação a ser empregada, podendo, por exemplo, ter sido utilizada a linguagem C.

3.4.2. Algoritmo da Busca Unidimensional

A busca unidimensional, que determina o escalar α_i , responsável pela minimização de $F(\mathbf{x}_i + \alpha_i \mathbf{p}_i)$ (equação (2.6.3-6)) é um algoritmo crítico, pois as propriedades de convergência e velocidade do algoritmo do gradiente conjugado dependem diretamente da mesma.

A correta determinação do passo ótimo α_i , à cada iteração, vai garantir que as direções de busca sejam realmente conjugadas, como pode ser visto nas equações (2.6.2-4) e (2.6.2-5) trans-

critas abaixo:

$$\frac{df}{d\alpha} (\mathbf{x}_1 + \alpha \mathbf{p}_1) \big|_{\alpha=\alpha_i} = F'(\mathbf{x}_1 + \alpha_i \mathbf{p}_1) \cdot \mathbf{p}_1 = 0 \quad (2.6.2-4)$$

$$\langle \mathbf{g}(\mathbf{x}_{i+1}), \mathbf{p}_1 \rangle = 0 \quad (2.6.2-5)$$

onde $\mathbf{g}(\mathbf{x}_{i+1})$ é o gradiente da função $F(\mathbf{x})$ no ponto \mathbf{x}_{i+1} .

Das equações acima, pode-se ver que as direções $\mathbf{g}(\mathbf{x}_{i+1})$ e \mathbf{p}_1 são conjugadas, desde que o escalar α_i seja ótimo a cada iteração, tornando o lado esquerdo de (2.6.2-4) igual a zero, ou seja, minimizando o funcional de custo a cada iteração. O fato de se ter direções conjugadas significa que uma função quadrática converge em n passos, onde n é a dimensão do espaço domínio do funcional.

Cada passo da busca envolve o cálculo do funcional de custo (equação (3.2.2-2) ou (3.3.2-9), no caso do motor DC), que é dado em função de $\mathbf{x}(t_f, u)$. Por sua vez, $\mathbf{x}(t_f, u)$ deve ser obtido por integração das equações de estado do sistema, de t_0 a t_f . Quanto mais refinado for o algoritmo da busca, mais valores da função objetivo $J[u]$ deverão ser calculados, tornando mais lenta a convergência. Por outro lado, ao se tentar reduzir o número de cálculos da função objetivo, através de uma busca mais simples, corre-se o risco dos valores de α_i , obtidos a cada iteração, não conduzirem a direções conjugadas, tornando também mais lenta a convergência.

A necessidade de se reduzir o número de cálculos da função objetivo, sem comprometer a convergência, impõe o uso de um algoritmo de busca unidimensional eficiente, uma vez que a velocidade de convergência do método do gradiente conjugado depende diretamente da busca unidimensional.

O algoritmo empregado para a busca unidimensional realiza a minimização da função objetivo por meio de extrapolação e interpolação, sendo uma combinação dos métodos de *Davies-Swann-Campey (DSC)* e de *Powell* [AOKI, 71] [HASDORFF, 76] [HIMMELBLAU, 72], com estimativa do passo inicial. A combinação dos métodos gera algoritmos mais eficientes que o uso de cada um separadamente.

A estimativa do passo inicial é feita considerando uma expansão em série de Taylor do funcional no ponto inicial, com termos de primeira ordem. Se for feita uma boa estimativa do passo inicial, não é necessário fazer mais que uma interpolação para se obter uma razoável aproximação do ponto de mínimo. Este é um fato importante, porque consegue-se uma apreciável redução do tempo de computação.

No algoritmo de *DSC*, partindo de um ponto inicial e de um passo inicial, procura-se a direção de decrescimento da função. Caminha-se, então, nessa direção, dobrando o passo a cada cálculo do valor da função até encontrar um valor maior que o valor do passo anterior, encontrando-se, assim, o intervalo (*bracket*) em que o mínimo está localizado. A seguir, determina-se três pontos igualmente espaçados pertencentes ao intervalo encontrado e interpola-se uma função quadrática para estimar o mínimo.

O algoritmo de *Powell* difere do método *DSC* por fazer sucessivas interpolações quadráticas a cada três pontos, independente de ter localizado o intervalo onde o mínimo está situado. Outra diferença é que no algoritmo *DSC* os pontos usados para interpolação estão igualmente espaçados, resultando em uma fórmula de interpolação quadrática mais simples, enquanto no de *Powell*, os pontos não são igualmente espaçados e a fórmula de interpolação quadrática é mais complexa.

O algoritmo implementado utiliza os procedimentos do método de *DSC* até encontrar o intervalo em que o mínimo está localizado. A partir desse ponto, utiliza os procedimentos de *Powell* para a determinação do ponto de mínimo.

3.4.3. Resultados Numéricos

Nesta seção, são apresentados os resultados da aplicação do método do gradiente conjugado para a obtenção do controle ótimo do motor DC, modelado na seção 3.3.2. Os parâmetros adotados para o motor são os da figura 3.2. O problema a ser resolvido é encontrar uma entrada de controle $u(t)$, tensão de campo do motor, que produza uma mudança em degrau de 10 radianos no eixo do motor.

Para a execução do programa, foi considerado $R = 5$ na equação (3.3.2-13), onde R é o termo que limita a energia da entrada de controle. As constantes de ponderação da equação (3.3.2-9), que têm por finalidade garantir que os valores do estado final sejam atingidos, foram escolhidas como $W_1 = W_2 = W_3 = 10000$.

Essas constantes reforçarão que o estado final seja:

- . posição do motor $x_1(t_f) = 10$ rads;
- . a velocidade $x_2(t_f) = 0$ rads/s e
- . a corrente de campo $x_3(t_f) = 0$ Amperes.

O controle é calculado no intervalo de tempo $[t_0, t_f] = [0, 5\text{seg}]$, com estimativa inicial $u_0(t) = 0$ volts. O programa foi executado com 150 pontos de discretização no intervalo de tempo considerado.

A tabela 3.1 mostra a convergência do quadrado da norma do gradiente e do valor final do funcional de custo a cada iteração. Essa tabela mostra que o funcional de custo atinge um valor constante a partir da 8ª iteração. Isso significa que, mesmo aumentando o número de iterações, a entrada de controle $u(t)$ resultante não vai se alterar, pois ela não consegue diminuir o valor do funcional de custo - o que indica que o mínimo do funcional foi atingido. Por outro lado, a norma do gradiente não diminui monotonicamente, como ocorre com o custo,

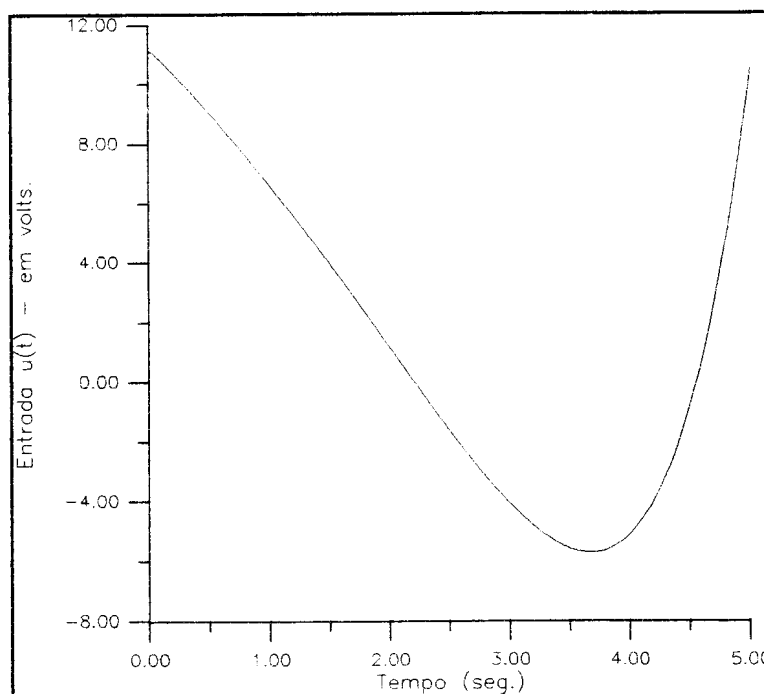


figura 3.3 - Entrada de controle $u(t)$ para o motor DC.

comportamento este característico dos métodos de gradiente [HAS-DORFF, 76].

TABELA 3.1 Convergência do Método do Gradiente Conjugado

Iteração	$ g ^2$	Custo
1	2.44E+8	4.896E+6
2	2.00E+8	4.06E+3
3	1.05E+5	1.86E+3
4	8.11E+6	1.76E+3
5	3.14E+6	1.11E+3
6	4.00E+5	1.08E+3
7	2.69E+6	1.04E+3
8	1.62E+5	9.69E+2
9	1.54E+3	9.68E+2
10	1.05E+3	9.68E+2
11	2.00E+4	9.68E+2
12	1.95E+3	9.67E+2

A figura 3.3 mostra a entrada de controle $u(t)$ no intervalo de tempo $[0, 5\text{seg}]$, obtida com 10 iterações do método do gradiente conjugado, para as condições definidas no início desta seção.

Pode-se observar que a tensão de campo para acelerar o motor inicialmente é positiva, de forma a levá-lo para a posição desejada. Em seguida, ela assume valores negativos, freiando o motor para evitar que o eixo ultrapasse a posição de 10 radianos. Considerando a dinâmica do sistema, a tensão volta a ficar positiva, de forma a anular a corrente de campo no tempo final $t_f = 5$ seg. Dessa forma, o comportamento da entrada de controle está conforme o esperado.

A figura 3.4 apresenta a resposta do sistema $x_1(t)$, obtida para a entrada de controle acima. Neste caso, o eixo atinge a posição de 10 radianos em $x(t_f) = 5$ seg. A curva tem a forma de um "s" invertido, devido às baixas velocidades próximas dos estados inicial e final.

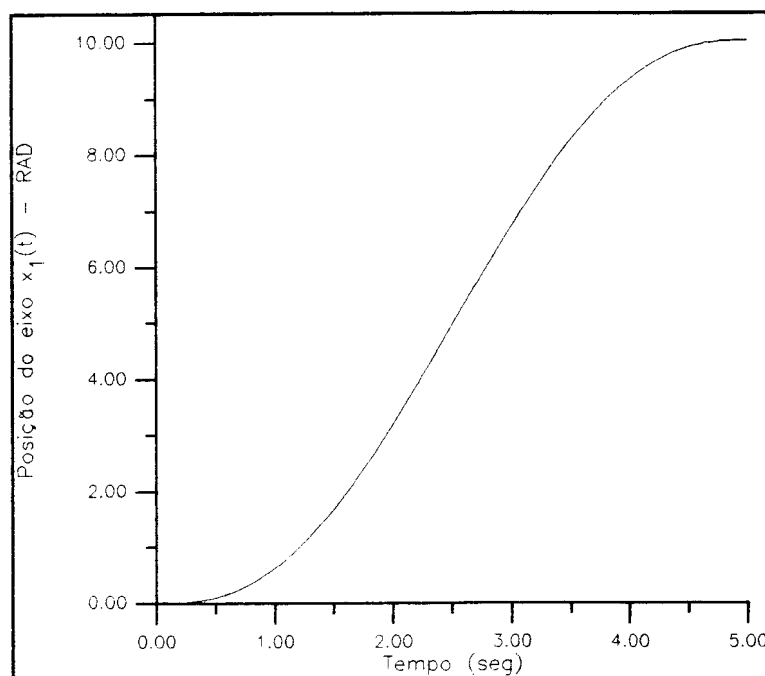


Figura 3.4 - Posição do eixo do motor DC.

O comportamento da corrente de campo e da velocidade angular do eixo do motor está ilustrado na figura 3.5. O eixo do motor parte do repouso, é acelerado até atingir um ponto de velocidade máxima. Em seguida, o mesmo é freiado até parar, alcançando o estado final. Durante este período, a corrente de campo atua no sentido de acelerar e freiar o eixo do motor, sendo que no estado final seu valor também é zero.

Esses resultados reproduzem o comportamento físico correto para o motor DC, segundo às condições impostas pelo problema, o que mostra que o modelamento do motor está correto e o método do gradiente conjugado se apresenta como uma excelente ferramenta para a solução de problemas de controle ótimo.

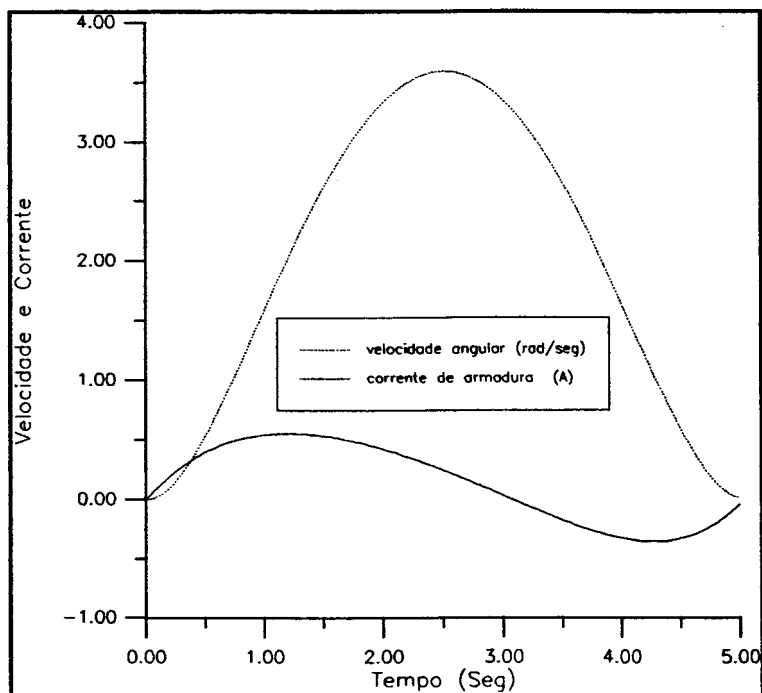


figura 3.5 - Velocidade do eixo do motor e corrente de campo.

Esta metodologia envolve a escolha adequada de parâmetros, tais como: as constantes de ponderação R e W_i , o número de pontos de discretização, a entrada de controle inicial $u_0(t)$ e o número de iterações do método. Na próxima seção, será realizada uma análise da sensibilidade do algoritmo em função desses parâmetros.

3.4.4. Análise de Sensitividade

Um aspecto importante para a aplicação do método do gradiente conjugado é a escolha adequada dos seguintes parâmetros: número de iterações e reinicializações do método; número de intervalos de discretização considerados no intervalo $[t_0, t_f]$; coeficientes de ponderação W_i e R ; e valores iniciais para a entrada de

controle $u_0(t)$. Procurando obter uma escolha ótima para esses parâmetros, será apresentada uma análise da sensibilidade do método em função de suas variações.

A análise será feita de forma autoconsistente, onde apenas um parâmetro sofre variação enquanto os demais permanecem inalterados. Nessa análise, os valores tomados como padrão para os parâmetros são: 10 iterações, nenhuma reinicialização, 150 intervalos de discretização, constantes de ponderação $W_1 = 10000$ e $u_0(t) = 0$ para a entrada de controle inicial.

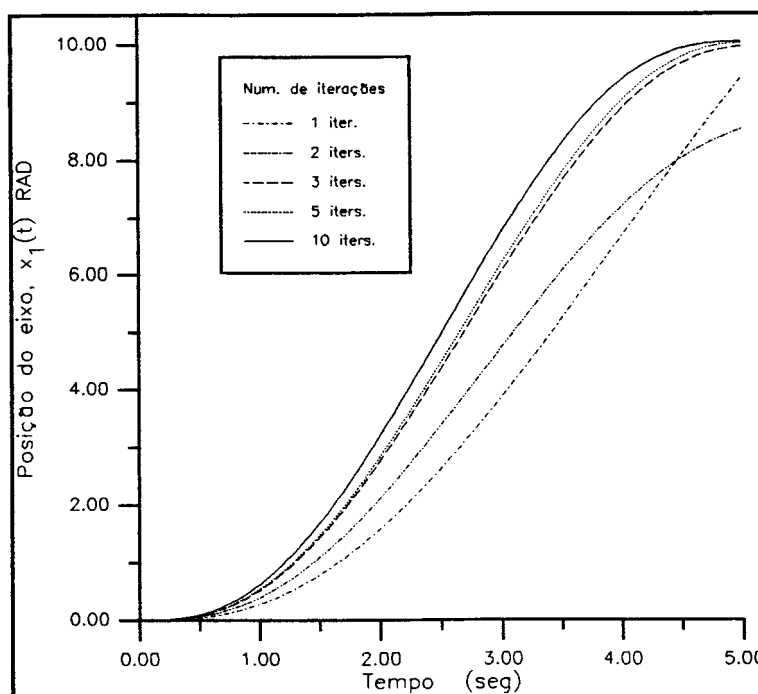


figura 3.6 - Influência do número de iterações sobre a resposta do sistema.

A figura 3.6 mostra a influência do número de iterações sobre a resposta do sistema. Os demais parâmetros permanecem constantes como já especificados. Nessa figura, pode-se observar que as diferenças mais significativas ocorrem nas primeiras três iterações, sendo que da terceira para a quinta iteração a resposta do sistema sofre pouca alteração.

Considerando que o tempo de computação com dez iterações é aproximadamente o dobro do tempo com cinco iterações e que a

diferença na resposta final, com dez e com cinco iterações, é pequena, então o algoritmo pode ser interrompido após a quinta iteração em aplicações onde há restrições no tempo de computação,

A sugestão de Fletcher e Reeves [Fletcher, 64] é confirmada na figura 3.6, devido às principais alterações na resposta do sistema ocorrerem antes da quinta iteração, lembrando, ainda, que o sistema global, como dado pelas equações (3.3.2-10) a (3.3.2-13), é de ordem 4. Como já visto no capítulo anterior, Fletcher e Reeves sugerem que a convergência do gradiente conjugado para um sistema de ordem n pode ser aumentada caso seja feita uma reinicialização do método a cada $n+1$.

A figura 3.7 mostra a posição do eixo do motor para 0, 1 e 5 reinicializações do método. Estas curvas foram traçadas com dados obtidos após 5 iterações, seguindo a sugestão de Fletcher e Reeves. As curvas para uma e cinco reinicializações são exatamente coincidentes, indicando que, neste caso, apenas uma reinicialização do método é suficiente para garantir a convergência.

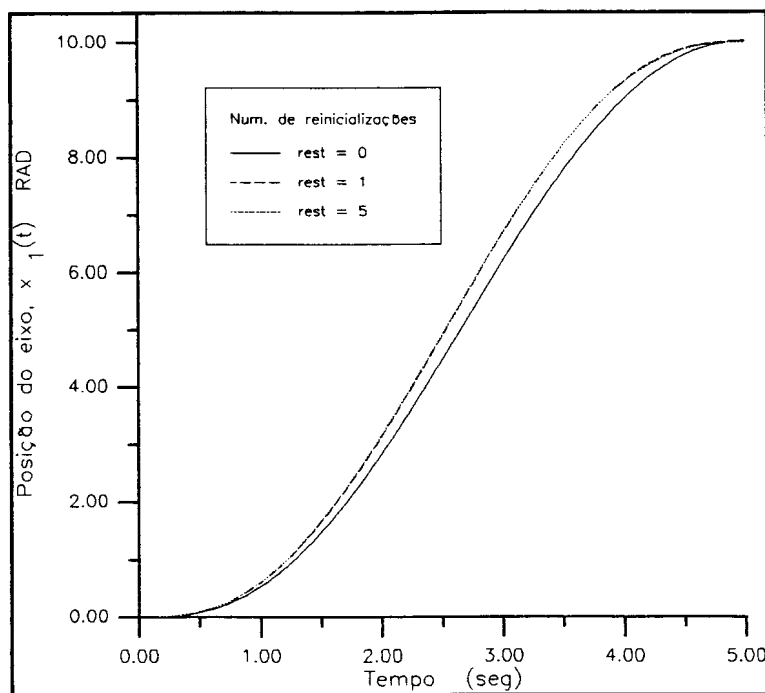
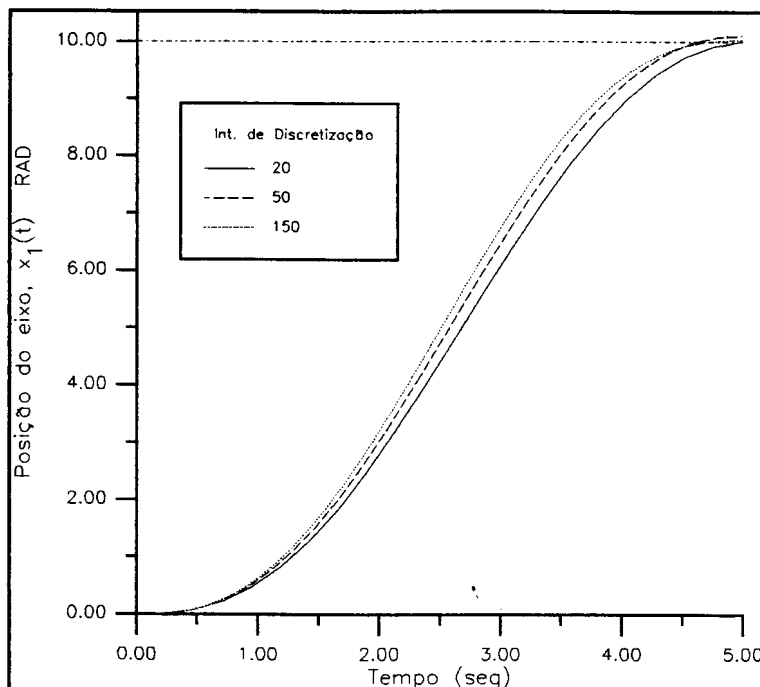


Figura 3.7 - Diferentes reinicializações.

A figura 3.7, a exemplo do caso anterior, vem confirmar que, em aplicações onde as restrições de tempo de execução são severas, o algoritmo pode ser interrompido com cinco iterações, sem a necessidade de reinicialização, reduzindo aproximadamente pela metade o tempo de computação.

Outro fator que afeta diretamente o tempo de computação é o número de intervalos de discretização entre t_0 e t_f . Isso ocorre porque cada ponto no intervalo de discretização, no procedimento de integração pelo método de Runge-Kutta, requer o produto de uma matriz de ordem $\underline{n} \times \underline{n}$.

O número de intervalos de discretização determina também a quantidade de memória necessária para o armazenamento dos dados. O número de componentes vetoriais que devem ser armazenados é dado por $(N+1) \cdot (2n+5)$, onde n é a dimensão do vetor de estado do sistema e N é o número de intervalos de discretização. Assim, um programa com $N=200$ requer 10 vezes mais memória para armazenamento dos vetores do sistema, além de ser 10 vezes mais lento do que outro com $n=20$.



Na figura 3.8

Figura 3.8 - Diferentes intervalos de discretização

são mostrados os resultados para 20, 50 e 150 intervalos de discretização entre t_0 e t_f . Pode-se observar que a resposta do sistema para $N=20$ não difere muito da resposta obtida com $N=150$. Caso haja restrições quanto ao tempo de processamento, $N=20$ pode ser uma boa escolha.

As funções de ponderação W_i definem o peso sobre os termos de penalização no funcional de custo. A finalidade desses termos é fazer com que o estado, no tempo final, seja plenamente atingido. De forma a verificar a influência dos mesmos sobre a resposta do sistema,

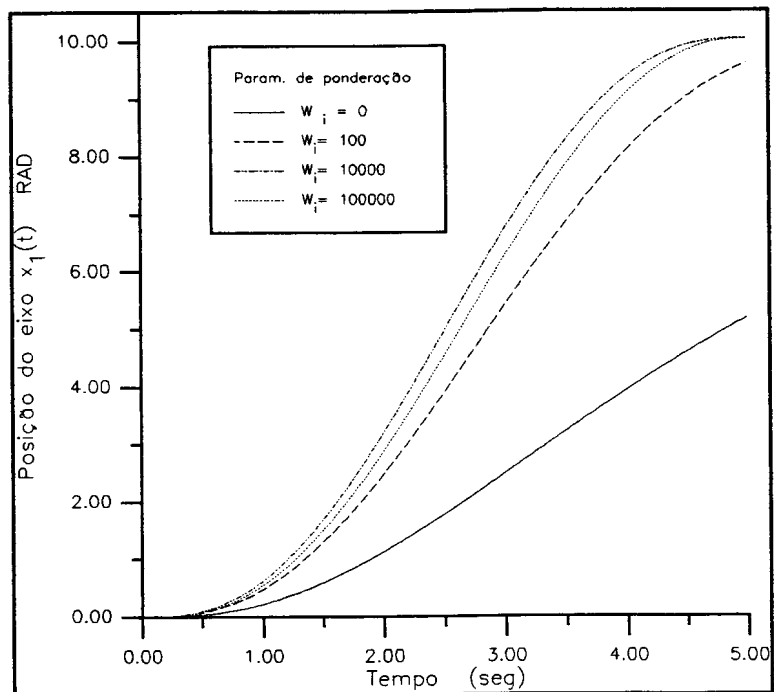


Figura 3.9 - Diferentes valores das funções de ponderação.

na figura 3.9 tem-se as curvas da posição do eixo para os valores de W_1 de 0, 100, 10000, e 100000. Pode-se observar que os valores $W_1 = 0$ e $W_1=100$ não são suficientes para garantir que o estado final desejado seja atingido ($x_1(t_f) = 10$ Rad, $x_2(t_f) = 0$ Rads/seg e $x_3(t_f) = 0$ Amp). Já os valores de $W_1 = 10000$ e $W_1 = 100000$ conseguem fazer com que o estado final seja atingido. Entretanto $W_1 = 10000$ leva a uma resposta melhor do que $W_1 = 100000$, porque este último valor, por ser muito grande, faz com que o termo devido ao erro de posição no funcional de custo tenha um peso relativo menor, piorando a resposta do sistema para valores iniciais do tempo (vide

equação (3.3.27)).

A influência dos valores iniciais das entradas de controle foi testada para os seguintes valores de $u_0(t)$: 0, 5, 50 e -5. A determinação desta influência é importante principalmente em sistemas em que não se tem conhecimento prévio das entradas de controle. A resposta para os valores de $u_0(t)$ = -5, 0 e 5 resultou em

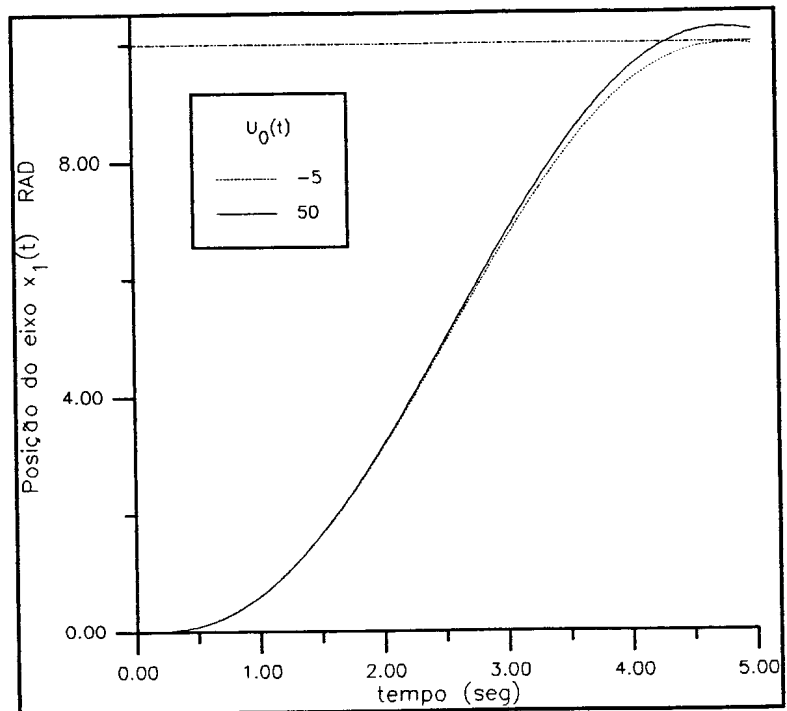


Figura 3.10 - Posição do eixo para diferentes valores de $u_0(t)$.

curvas bem próximas no gráfico da figura 3.10, de modo que somente os dois valores mais extremos de $u_0(t)$ (-5 e 50) foram mostrados. Nesta figura pode-se observar que para $u_0(t) = 50$ a curva ultrapassa o valor de 10 Rads, devido ao excesso de energia aplicado inicialmente, que resultou em energia residual final para a entrada de controle. As curvas das entradas de controle resultantes para $u_0(t) = -5, 0$ e 50 estão mostradas na figura 3.11, onde pode ser observado o excesso de energia mencionado para $u_0(t)$.

Em situações práticas, onde não se tem inicialmente uma estimativa do valor das entradas de controle, é uma boa prática estabelecer $u_0(t)=0$. Desta maneira, evita-se introduzir uma perturbação inicial no sistema, que pode-se propagar até a iteração

final, como ocorreu com $u_0(t)=50$.

Dos casos já estudados, pode-se perceber que o tempo de computação depende, principalmente, do número de iterações, de reinicializações e do número de intervalos de discretização. A variação independente de cada parâmetro já foi estudada, porém é interessante ver o que ocorre quando a variação destes parâmetros se dá simultaneamente.

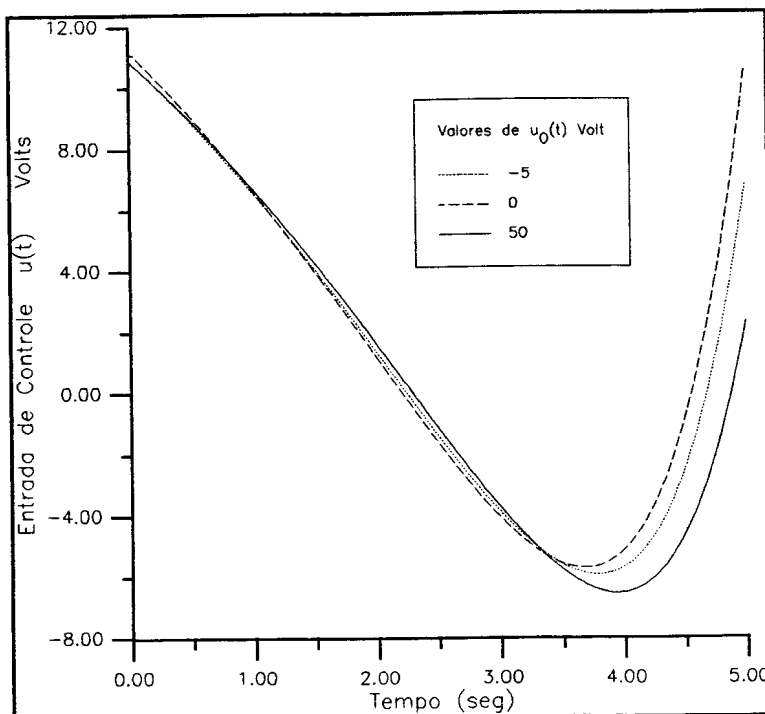


Figura 3.11 - Entradas de controle para diferentes valores $u_0(t)$.

Desta forma, foram escolhidos dois casos de teste:

- caso A, onde procurou-se obter a maior precisão do método e, conseqüentemente, o maior tempo de computação;
- caso B, onde a precisão foi preterida em função da maior velocidade de computação.

No caso A, foram utilizados 150 intervalos de discreti-

zação, 10 iterações e uma reinicialização do método. No caso B, foram utilizados 20 intervalos de discretização, 5 iterações e nenhuma reinicialização do método.

O tempo de computação para execução do caso A é cerca de 30 vezes maior que o tempo para execução do caso B.

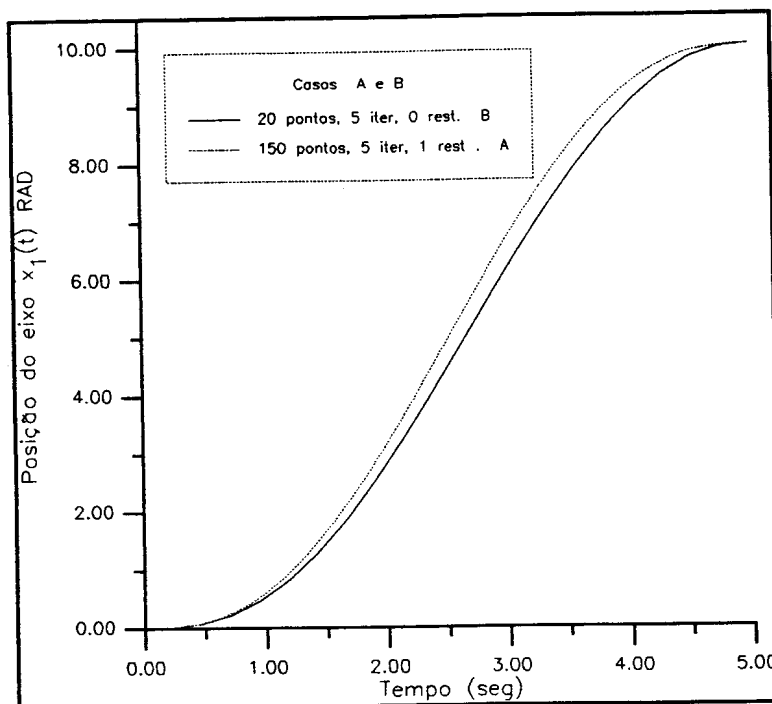


figura 3.12 - Posição do eixo para os casos A e B.

A figura 3.12 mostra a resposta do eixo do motor para estes dois casos. Observa-se, assim, que a melhor resposta do caso A pode não compensar quando se leva em conta a diferença entre os tempos de computação. Em aplicações de tempo real, onde as restrições de tempo são severas, a resposta do caso B pode ser aceitável.

CAPÍTULO 4

Extensão do Método do Gradiente Conjugado para Controle Ótimo em Tempo Real

4.1. Introdução

O método do gradiente conjugado, apresentado no capítulo anterior, se apresenta como uma boa ferramenta para a obtenção do controle ótimo de processos genéricos. Ao contrário de outros métodos numéricos, ele não requer um tempo de processamento muito grande e pode ser aplicado a sistemas não lineares e com restrições, tanto nas variáveis de estado quanto de controle.

A aplicação do método do gradiente conjugado, entretanto, deve se dar entre dois instantes de tempo fixados *a priori*, ou seja, o tempo final deve ser conhecido antes do início dos cálculos - fato este que dificulta sua aplicação no controle ótimo de processos em tempo real.

Para contornar essa dificuldade, optou-se pela dilatação do intervalo de tempo de operação do sistema para a aplicação do método do gradiente conjugado. Desse modo, este capítulo apresenta a análise de comportamento do método quando o intervalo de operação do sistema a ser controlado é muito maior do que a constante de tempo dominante do mesmo.

Para solucionar a dificuldade mencionada, no entanto, uma extensão do método do gradiente conjugado é apresentada, de modo que ele possa ser empregado em sistemas onde o tempo final não é fixado. Essa extensão será aplicada na resolução de um problema em tempo real. Para tornar as restrições de tempo de computação menos severas, foi empregado um esquema tipo linha de montagem ("pipeline").

4.2. Sistemas de Controle em Tempo Real

Na área de controle de processos, algumas vezes os termos "controle *on-line*", "controle digital direto" e "controle em tempo real" são utilizados indistintamente, para denominar um tipo de sistema de controle de processos, em que o computador calcula e envia as entradas de controle a um processo, durante a operação do mesmo. Para este trabalho, no entanto, foi selecionado o termo "controle em tempo real", por se acreditar ser o mais adequado, face as explicações que seguem.



O termo on-line, significando "em linha" ou "conectado", é empregado em computação para designar quando um periférico, um arquivo ou um recurso genérico do computador está sob o controle direto de um programa, com o sentido de que pode ser usado imediatamente. Esse termo é usado em contraposição a *off-line*, que quer dizer "desconectado" ou "fora de linha". Este último termo aplica-se a dispositivos e arquivos que não estão diretamente conectados ao computador, de forma que os programas não têm acesso a eles [CLULEY, 75].

Os primeiros computadores operavam apenas no modo *off-line*, onde os resultados dos cálculos eram armazenados em fita perfurada, magnética ou em cartões perfurados, ou ainda, listados em formulários contínuos, para posteriormente serem analisados ou processados por outros computadores ou programas.

Em controle de processos, ainda se usa o modo desconectado, principalmente nos casos em que o computador é empregado como ferramenta para a análise e projeto de sistemas de controle. Neste trabalho, para todos os casos de aplicação do método do gradiente conjugado vistos até aqui foi usado o modo desconectado. Nesses casos, os resultados obtidos eram armazenados em disco, para posterior análise e elaboração de gráficos.

O termo "controle digital direto" é empregado, normalmente, para designar um sistema de controle, em que um ou mais controladores analógicos foram substituídos pelo computador [BENNETT, 90]. Os controladores utilizados nesses processos geralmente são de três modos, ou seja, do tipo Proporcional mais Integral

mais Derivativo (PID). Nesse caso, o computador fornece as entradas de controle para o processo, no lugar dos controladores analógicos. Quando o número de controladores substituídos é elevado, o controle digital direto apresenta um custo menor. Para cada controlador, tanto os sinais de erro provenientes do processo como os sinais de controle gerados pelo computador são multiplexados, para que um único computador possa substituir de fato os vários controladores. Esse tipo de controle deve ser realizado em tempo real, ou seja, a ação de controle deve ser calculada durante a operação do sistema, o que, normalmente é possível, uma vez que os algoritmos de controle PID são relativamente simples.

Quando o "controle digital direto" é calculado para uma lei de controle mais elaborada, como o controle ótimo, nem sempre o computador é suficientemente rápido para que este cálculo possa ser realizado em tempo real. Nesse caso, utilizando-se as técnicas de computação paralela ou de computação distribuída, vários processadores podem ser empregados na implementação do controlador.

Aplicações de computação distribuída e paralela no controle de processos em tempo real podem ser encontradas em [BENNETT, 90], nos capítulos 6 e 7, respectivamente. Nessas aplicações, bem como na maioria dos casos de aplicações de computação paralela no controle ótimo em tempo real de processos, a arquitetura empregada é fortemente dependente do processo em questão, impedindo sua aplicação em outros tipos de processos.

Bottura e colaboradores [BOTTURA, 77] apresentaram um

caso de controle digital direto para o controle ótimo de um sistema linear com custo quadrático, onde o processo foi simulado na parte analógica de um computador híbrido e o controle ótimo foi calculado na parte digital. Entretanto, para que a parte digital tivesse tempo para realizar o cálculo do controle ótimo, foram utilizados os recursos de "hold" do computador analógico para "congelar" a simulação do sistema a ser controlado. Observa-se que esse caso não pode ser considerado como um sistema de controle em tempo real, e sim como uma simulação de tal sistema.

O estabelecimento de uma definição precisa de sistemas de tempo real poderia facilitar a classificação dos sistemas de controle. Stankovic [STANKOVIC, 90] apresenta a seguinte definição:

"um sistema de computação em tempo real é aquele em que a precisão do sistema não depende apenas dos resultados lógicos da computação, mas também do instante de tempo em que os resultados são produzidos".

Segundo essa definição, um sistema de computação de reservas de passagens aéreas e um sistema bancário de caixa eletrônico não seriam sistemas de tempo real, mas sim sistemas "on-line".

Uma definição mais abrangente pode ser encontrada em Magalhães [MAGALHÃES, 90][AUDSLEY, 94], onde os sistemas de tempo real são divididos em dois grupos. No primeiro estão os sistemas de tempo real com restrições brandas de tempo (*soft real time*),

que, embora exijam tempo de resposta rápido, não sofrem restrições temporais em tempo de execução. No segundo grupo, por outro lado, estão os sistemas de tempo real com restrições severas de tempo (*hard real time*), cuja correção do resultado depende não somente do resultado lógico mas também dos instantes em que o resultado é produzido.

De acordo com esta última definição, os sistemas de reservas de passagens aéreas e os sistemas bancários de caixa eletrônico podem ser considerados sistemas de tempo real, só que sistemas com restrições brandas de tempo de processamento.

Alguns autores que tratam de metodologias de desenvolvimento de sistemas de tempo real, do ponto de vista da engenharia de programação (*software engineering*), como [PRESSMAN, 87] e [HATLEY, 90], simplesmente apresentam o termo "sistema de tempo real" de modo intuitivo.

Face aos conceitos expostos, é que se pode concluir que nos sistemas de controle, bem como neste trabalho, deve sempre ser considerada a definição de sistemas de tempo real com restrições severas de tempo, de [MAGALHÃES, 90]. Isso porque, quando se usam os termos "controle digital direto" de processos ou sistema de "controle em tempo real", significa que o computador está calculando e enviando as entradas de controle para o processo durante sua operação, mantendo, simultaneamente, as restrições temporais impostas pela dinâmica do sistema.

4.3. Considerações sobre o Tempo Final de Sistemas em Tempo Real

Quando se fala em controle em tempo real de um processo, nem sempre o intervalo de tempo de operação deste processo é definido. Por exemplo, em plantas industriais, como refinarias de petróleo, siderúrgicas e indústrias químicas, os processos são operados de forma contínua por vários dias e até mesmo meses. Em outros casos, como os sistemas de controle de aeronaves e mísseis, o tempo de operação não é tão longo, mas também é indeterminado. Em outras palavras, o que normalmente se espera de um sistema de controle em tempo real é que, uma vez iniciada sua operação, esta continue até que o operador decida pela parada ou desligamento do sistema.

Assim sendo, o sistema de controle deve ser capaz de operar com o tempo final livre (não fixado), ou seja, o intervalo em que o controle ótimo vai ser determinado é $[t_0, t]$, onde t é não fixado. Assim, para qualquer instante $t > t_0$, o controle ótimo minimiza o funcional de t_0 até o instante t .

Considerando que o objetivo deste capítulo é aplicar o método do gradiente conjugado à solução de problemas de controle ótimo em tempo real, o método deve ser estendido para trabalhar no intervalo $[t_0, t]$, onde t pode assumir qualquer valor, em particular $t \gg t_0$.

Uma possível forma de cobrir esse intervalo de tempo

$[t_0, t]$ no método do gradiente conjugado original (capítulo 3), é considerar simplesmente que o instante final t_f pode assumir qualquer valor, inclusive um valor tal que $t_f \gg t_0$, caso este abordado na próxima seção.

4.4. Método do Gradiente Conjugado com Dilação do Intervalo de Tempo de Operação

Segundo o estabelecido na seção anterior, o controle ótimo em tempo real de um processo requer que as entradas de controle sejam calculadas durante todo o tempo de operação do sistema.

No capítulo anterior, foi mostrado que a obtenção do controle ótimo pelo método do gradiente conjugado recai em um problema de otimização com tempo final t_f fixo, pois o funcional de custo $J[\mathbf{u}] = \phi(\mathbf{x}(t_f, \mathbf{u}))$ é uma função do tempo final.

De forma a investigar a possibilidade de aplicação do método do gradiente conjugado em um problema com o tempo final livre, foi executado o mesmo caso do motor DC (seção 3.3), alterando-se, porém, o intervalo de operação do sistema, que convencionaremos a chamar de Δt_{op} . Dessa forma, foi considerado o intervalo de $[0, 50]$ seg, onde $t_0 = 0$, $t_f = 50$ seg e $\Delta t_{op} = 50$ seg, enquanto que no caso de teste do capítulo 3 tem-se que $\Delta t_{op} = 5$ seg (seção 3.4.3), configurando uma dilatação no intervalo de tempo de operação do sistema, de 10 vezes.

O restante dos parâmetros escolhidos para a execução do programa foram basicamente os mesmos da seção 3.4.3 (150 intervalos de discretização; $u_0(t) = 0$ volts; $R = 5$; $W_i = 10000$), com exceção de que o número de iterações do método do gradiente conjugado passou a ser 5, com reinicialização do mesmo. Essas alterações seguem a sugestão de Fletcher e Reeves (ver seção 3.4.4), fazendo com que, neste caso, o tempo de processamento seja praticamente o mesmo que no caso anterior (10 iterações sem reinicialização do método), apresentando, entretanto, a vantagem de facilitar a convergência.

Como resultado, a posição do eixo do motor em função do tempo $x_1(t)$ e a entrada de controle $u(t)$ estão mostradas nas figuras 4.1 e 4.2, respectivamente.

Comparando a figura 4.1 com a figura 3.4, pode-se observar que, no presente caso, a resposta do sistema é muito mais lenta que no caso anterior. Na figura 3.4, em apenas 4 segundos o motor já está praticamente na posição desejada de 10 radianos, ao passo que, no sistema da figura 4.1, só atinge os 10 radianos após

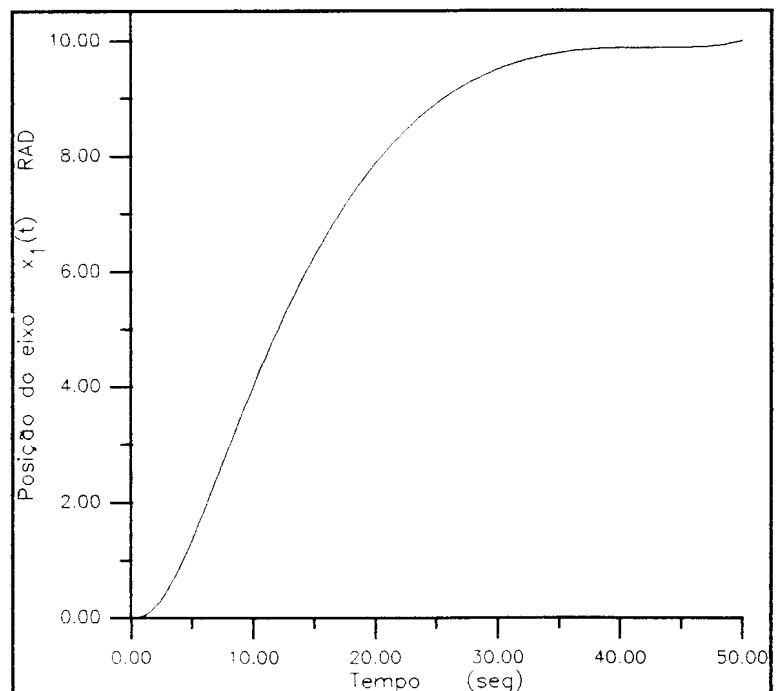


Figura 4.1 - Posição do eixo do motor para $\Delta t_{op} = 50\text{seg}$.

35 segundos.

Quanto a entrada de controle, $u(t)$, pode-se observar na figura 4.2 que o módulo da tensão de entrada não ultrapassa 0,6 volts, e que a parte negativa da curva é quase zero, significando que o motor praticamente não precisa ser freiado.

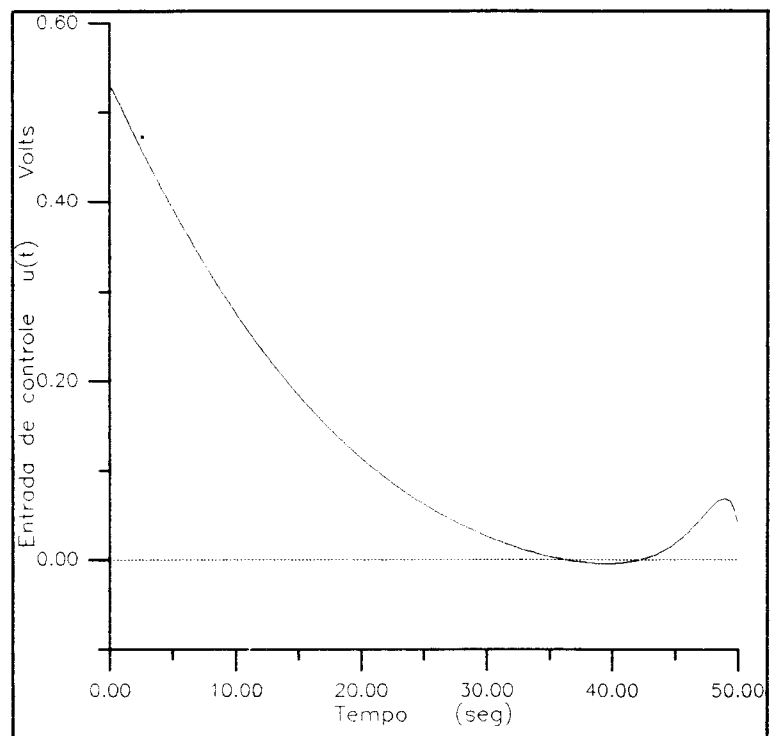


Figura 4.2 - Entrada de controle $u(t)$ para $\Delta t_{op} = 50\text{seg.}$

Comparando esta figura com a figura 3.3, verifica-se que quando $\Delta t_{op} = 5\text{seg}$, a entrada de controle oscila entre +12 e -6 volts.

As entradas de controle para ambos os casos estão fisicamente coerentes, pois, no caso do capítulo 3, o intervalo de tempo é 10 vezes menor que no presente caso, logo a energia que deve ser aplicada no motor deve ser maior - o que explica a diferença entre as amplitudes nos sinais de controle. A parte negativa na entrada de controle da figura 3.3 significa que o motor deve ser freiado após a aplicação da tensão inicial, para que ele possa chegar no estado final $x(t_f)$ com velocidade zero.

Para maior facilidade de comparação, os gráficos das figuras 3.4 e 4.1 foram sobrepostos na figura 4.3, e os das figuras 3.3 e 4.2 foram reproduzidos na figura 4.4.

Da figura 4.3 pode-se observar que a resposta do sistema para $\Delta t_{op} = 50\text{seg}$ é ineficiente, se comparada à resposta para $\Delta t_{op} = 5\text{seg}$. A resposta para 50 segundos é muito lenta, devido à pouca energia da entrada de controle obtida para este caso.

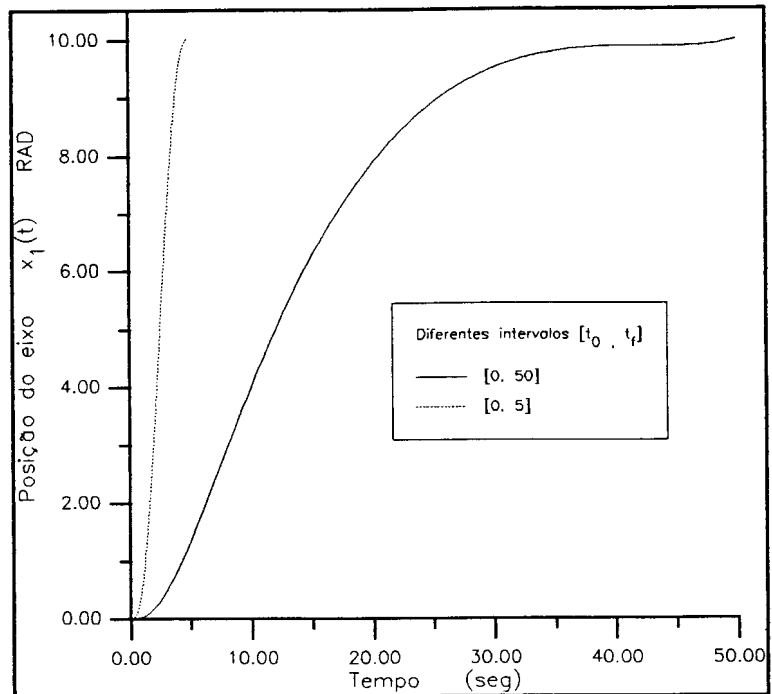


Figura 4.3 - Sobreposição da resposta do sistema para $\Delta t_{op} = 5$ e 50 seg.

4.4.1. Análise do Resultado

Considerando que o resultado obtido para $\Delta t_{op} = 50\text{seg}$ é qualitativamente muito pobre, é interessante determinar suas causas.

Uma primeira hipótese seria atribuir este desempenho insatisfatório ao erro de integração no método de Runge-kutta, uma vez que o passo da integração aumenta proporcionalmente ao aumento do Δt_{op} em que o controle é calculado. Por exemplo, com 150 inter-

valos de discretização, para $\Delta t_{op} = 5\text{seg}$, tem-se $5/150 = 0,0333\text{seg}$ para o passo do Runge-kutta; para $\Delta t_{op} = 50\text{seg}$, o passo do Runge-Kutta também aumenta 10 vezes, indo para $0,333\text{seg}$.

Entretanto, na seção 3.4.4, foi realizada uma análise de sensibilidade do método para diferentes intervalos

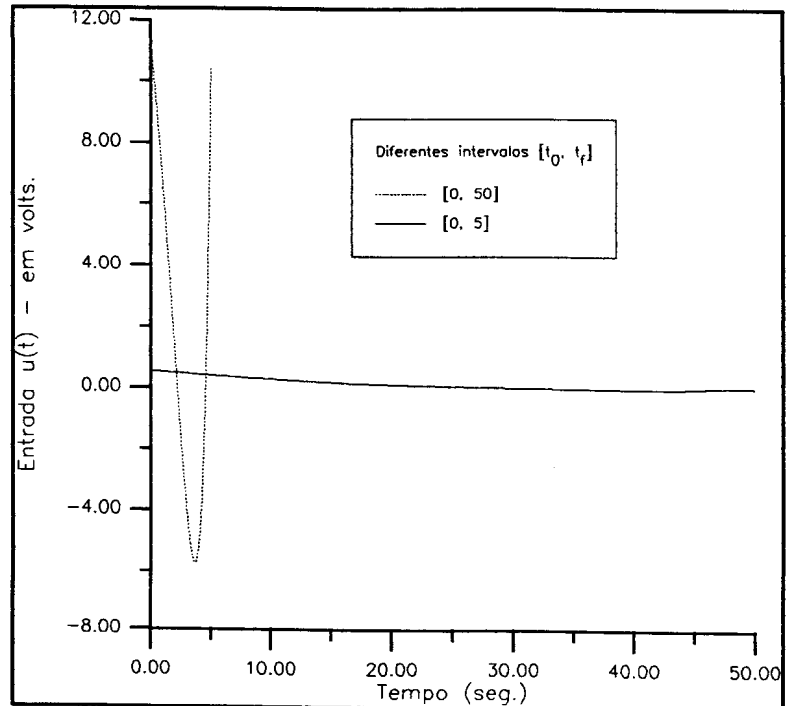


Figura 4.4 - Sobreposição das entradas de controle para $\Delta t_{op} = 5$ e 50seg .

los de discretização, cujos resultados estão mostrados na figura 3.8. Analisando a figura, pode-se observar que a curva de resposta do eixo do motor com 20 intervalos de discretização difere muito pouco da curva com 150 intervalos. Nesse caso, o passo do Runge-Kutta é $5/20 = 0,25\text{seg}$, valor este bem próximo do passo igual a $0,333\text{seg}$ para $\Delta t_{op} = 50\text{seg}$. Assim sendo, observa-se que o fraco desempenho obtido com $\Delta t_{op} = 50\text{seg}$ não se deve ao passo de integração empregado no método de Runge-Kutta.

A figura 4.4 mostra que a entrada de controle para $\Delta t_{op} = 50\text{seg}$ apresenta uma pequena amplitude. Uma forma de tentar corrigir este problema seria alterar o valor da constante de ponderação R . Como já foi estabelecido na seção 3.3.2, o termo R foi acrescentado no funcional de custo para limitar a amplitude da entrada de controle e evitar que a mesma atinja valores que não possam ser

valos de discretização, para $\Delta t_{op} = 5\text{seg}$, tem-se $5/150 = 0,0333\text{seg}$ para o passo do Runge-kutta; para $\Delta t_{op} = 50\text{seg}$, o passo do Runge-Kutta também aumenta 10 vezes, indo para $0,333\text{seg}$.

Entretanto, na seção 3.4.4, foi realizada uma análise de sensibilidade do método para diferentes intervalos de discretização, cujos resultados estão mostrados na figura 3.8.

Analizando a figura, pode-se observar que a curva de resposta do eixo do motor com 20 intervalos de discretização difere muito pouco da curva com 150 intervalos. Nesse caso, o passo do Runge-Kutta é $5/20 = 0,25\text{seg}$, valor este bem próximo do passo igual a $0,333\text{seg}$ para $\Delta t_{op} = 50\text{seg}$. Assim sendo, observa-se que o fraco desempenho obtido com $\Delta t_{op} = 50\text{seg}$ não se deve ao passo de integração empregado no método de Runge-Kutta.

A figura 4.4 mostra que a entrada de controle para $\Delta t_{op} = 50\text{seg}$ apresenta uma pequena amplitude. Uma forma de tentar corrigir este problema seria alterar o valor da constante de ponderação R . Como já foi estabelecido na seção 3.3.2, o termo R foi acrescentado no funcional de custo para limitar a amplitude da entrada de controle e evitar que a mesma atinja valores que não possam ser

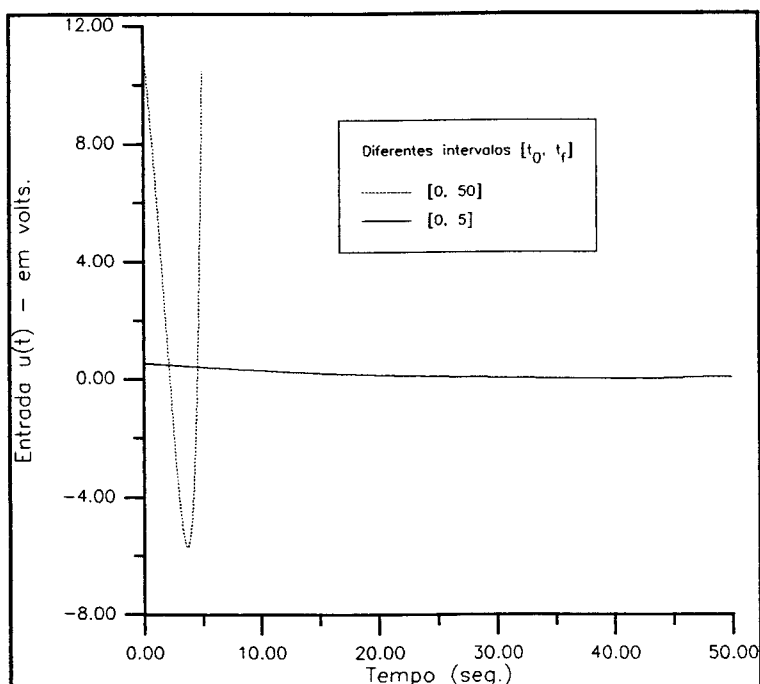


Figura 4.4 - Sobreposição das entradas de controle para $\Delta t_{op} = 5$ e 50seg .

Observando-se as entradas de controle para o sistema na figura 4.6, pode-se notar que nenhuma das curvas de controle ultrapassa o valor de $0,7$ volts, mesmo com $R = 0$, ao passo que, pa-

ra $\Delta t_{op} = 5\text{seg}$, na figura 3.3, a amplitude máxima das entradas de controle ultrapassa os 11 volts.

Face ao exposto, pode-se concluir que para $\Delta t_{op} = 50\text{seg}$, o parâmetro R não influencia de maneira significativa as entradas de controle, nem a resposta do sistema.

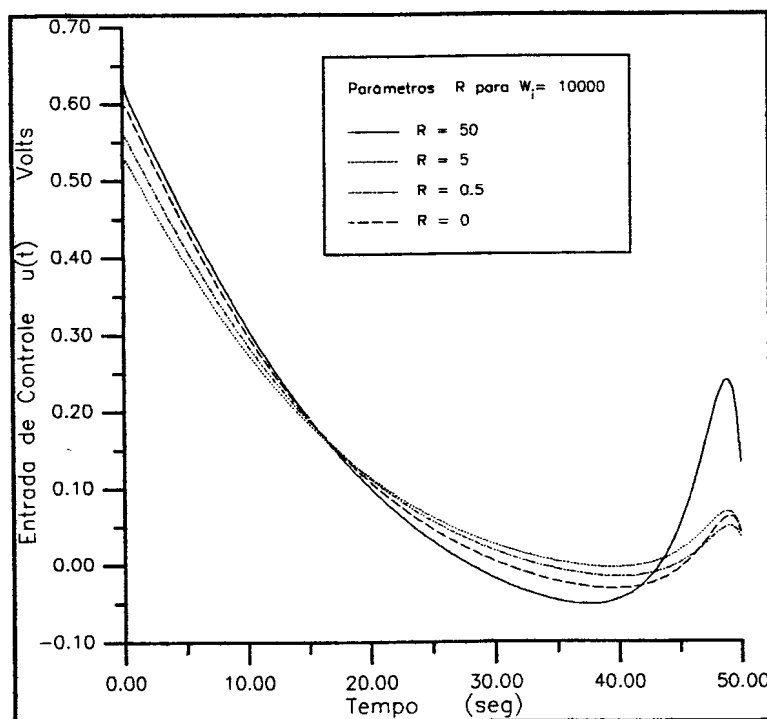


Figura 4.6 - Entradas de controle para $\Delta t_{op} = 50\text{seg}$ e diferentes valores de R .

Os parâmetros de ponderação W_i no funcional de custo do sistema têm por finalidade garantir que o estado final seja atingido, como pode ser visto na equação (3.3.2-7). Entretanto, dependendo dos valores que estes parâmetros assumem, pode ocorrer um mascaramento dos outros parâmetros do funcional de custo. Para verificar esta possibilidade, o programa foi executado para diferentes valores de W_i . A resposta do sistema está ilustrada na figura 4.7 e as respectivas entradas de controle na figura 4.8.

Na figura 4.7, pode-se observar que a melhor resposta é realmente obtida com $W_i = 10000$. Valores inferiores a este causam sobre-elevação, piorando a resposta do sistema; valores superiores, como $W_i = 100000$, tornam a resposta do sistema ainda mais lenta.

As curvas para as entradas de controle mostram que

quanto menor o valor de W_i , maior é a energia aplicada ao sistema. No entanto, o maior valor para entrada de controle é +2,5 volts, que está bem distante dos quase 12 volts obtidos quando $\Delta t_{op} = 5\text{seg}$. Assim, a variação de W_i não altera significativamente a resposta do sistema quando este opera com $\Delta t_{op} = 50\text{seg}$.

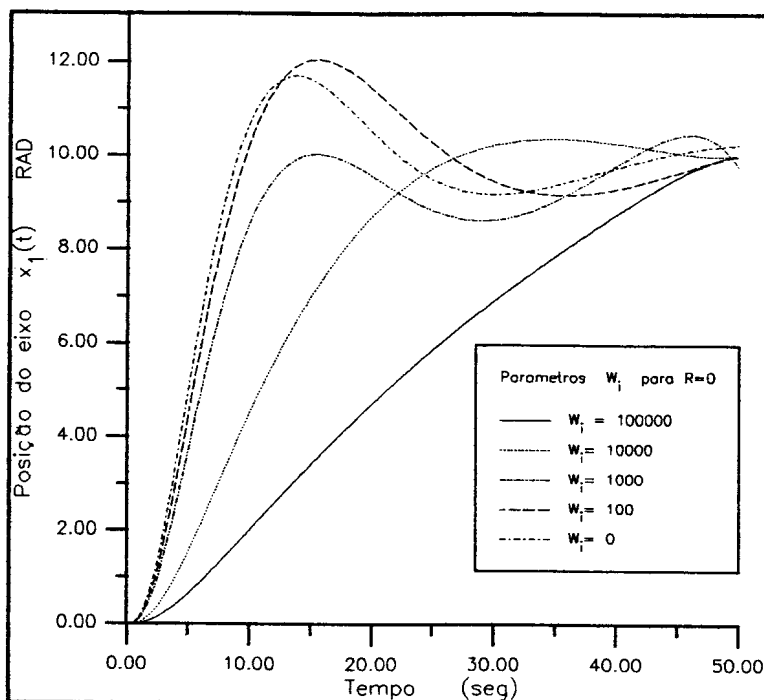


Figura 4.7 - Posição do eixo para $\Delta t_{op} = 50\text{seg}$ e diferentes W_i .

Considerando que todos os parâmetros que poderiam influenciar a resposta do eixo do motor foram analisados e que nenhum apresentou melhoria significativa na resposta do sistema, pode-se concluir que o método do gradiente conjugado, quando aplicado a este problema de teste, não apresenta resultados satisfatórios para

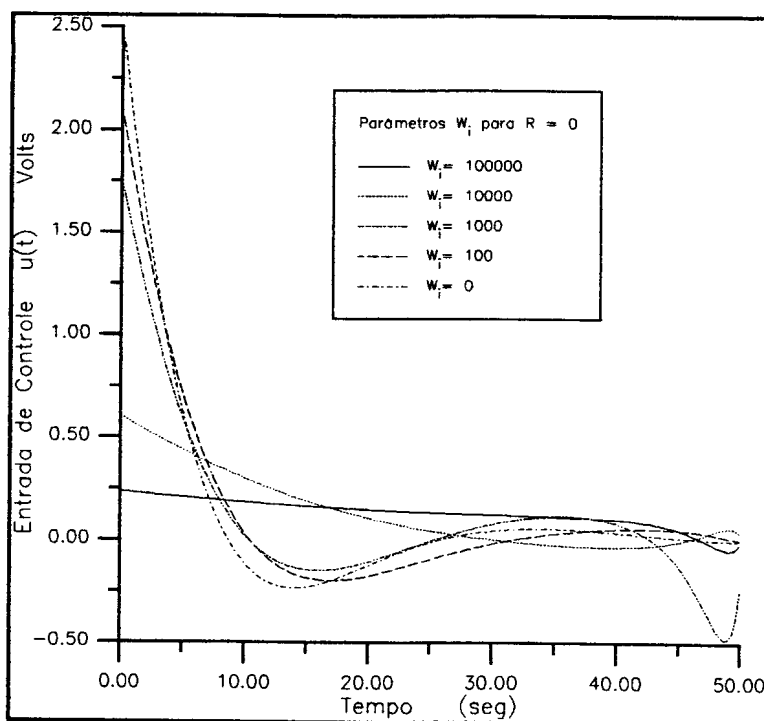


Figura 4.8 - Entradas de controle para $\Delta t_{op} = 50\text{seg}$ e diferentes W_i .

$$\Delta t_{op} = 50 \text{seg.}$$

4.4.2. Constante de Tempo Dominante para o Motor DC

As equações de estado para o motor DC são definidas pelas equações (3.3.2-10) a (3.3.2-13), onde as equações (3.3.2-10) a (3.3.2-12) definem a dinâmica do sistema e a equação (3.3.2-13) foi adicionada para envolver o termo integrável do funcional de custo. Para maior facilidade de leitura, as equações referentes à dinâmica do sistema são transcritas a seguir:

$$\dot{x}_1 = x_2 \quad (3.3.2-10)$$

$$\dot{x}_2 = -\frac{x_2}{3} + 5x_3 \quad (3.3.2-11)$$

$$\dot{x}_3 = -x_3 + 0,1u \quad (3.3.2-12)$$

Estas equações geram um sistema de equações diferenciais lineares de primeira ordem, a coeficientes constantes, do tipo:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (4.4.2-1)$$

com

$$\mathbf{x}(0) = \mathbf{0} \quad (4.4.2-2)$$

onde a matriz **A** é dada por:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{1}{3} & 5 \\ 0 & 0 & -1 \end{bmatrix} \quad (4.4.2-3)$$

Na equação (4.4.2-1), quando $\mathbf{u}(t) = \mathbf{0}$, resta somente a parte homogênea da equação diferencial, a qual está relacionada com as constantes de tempo do sistema. A parte não homogênea, por sua vez, está relacionada com os termos forçantes (entradas do sistema).

Considerando que os autovalores da matriz **A** são 0, -1/3 e -1, as constantes de tempo características para o motor DC são, respectivamente, 3 e 1 seg, uma vez que estas constantes são dadas pelo inverso dos autovalores da matriz **A** [CHEN, 70], gerando soluções transitórias do tipo $e^{-t/3}$ e e^{-t} . Estes valores mostram que o modelo do motor DC (seção 3.3.2) é estável, ressaltando que a constante de tempo dominante, ou a constante de tempo de maior valor, é 3seg.

É interessante ver o que ocorre quando se tenta controlar o sistema com um Δt_{op} menor que a constante de tempo dominante. Para isso, o método do gradiente conjugado foi executado para calcular as entradas de controle para o motor DC com $\Delta t_{op} = 1$ seg, ou seja, $t_0 = 0$ e $t_f = 1$ seg. Os demais parâmetros foram mantidos iguais aos casos anteriores ($u_0(t)=0$, 150 intervalos de discretização, $R = 5$, $W_i = 10000$ e 5 iterações do método com 1 reini-

cialização).

Da resposta do sistema para este caso, figura 4.9, observa-se que o estado final atinge 5,2 radianos, o que é pouco mais da metade do valor desejado para a posição final do eixo (10 radianos). Entretanto, observando-se a figura 3.4, que mostra a resposta do eixo para $\Delta t_{op} = 5\text{seg}$, pode-se notar que, para $t = 1\text{seg}$, a posição do eixo é aproximadamente 0,5 radianos. Logo, a resposta para o presente caso é muito mais rápida, embora não chegue aos 10 radianos.

A partir da figura 4.10, onde está a entrada de controle para o presente caso, pode-se observar que a potência

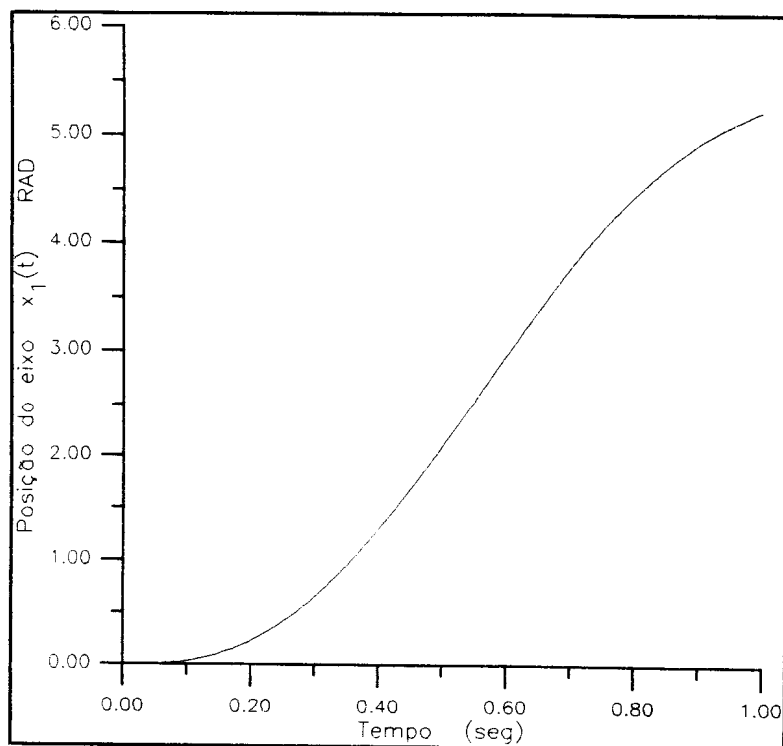


Figura 4.9 - Posição do eixo do motor para $\Delta t_{op} = 1\text{seg}$.

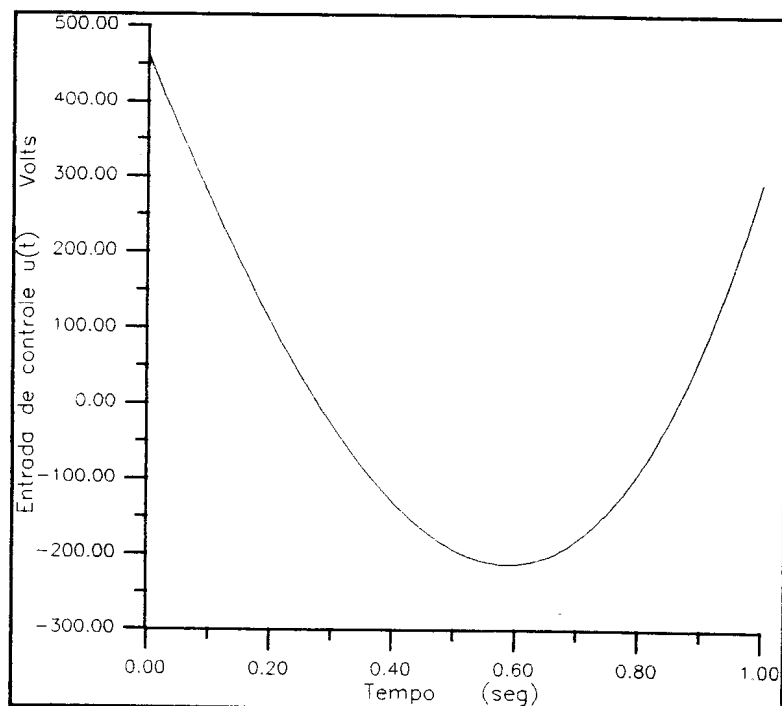


Figura 4.10 - Entrada de controle para o motor DC com $\Delta t_{op} = 1\text{seg}$.

despendida é muito maior que no caso da figura 3.3 ($\Delta t_{op} = 5\text{seg}$). Considerando que na figura 4.10 a amplitude máxima é, aproximadamente, 450 volts e que, no caso da figura 3.3, aproximadamente 12 volts, a relação entre as potências é $450/12 = 1406,25$. Portanto, a potência máxima instantânea aplicada no motor é, aproximadamente, 1400 vezes maior do que quando o controle é calculado para $\Delta t_{op} = 5 \text{ seg}$.

No modelo teórico, o valor atribuído a R poderia ser zero, de modo a tirar o efeito da penalização sobre a tensão de campo no funcional de custo do sistema (equação (3.3.2-7)). Dessa forma, valores ainda maiores da tensão de campo seriam obtidos, melhorando a resposta do sistema. Entretanto, na prática, nem mesmo a tensão de 450 volts poderia ser aplicada ao campo do motor, pois poderia acarretar a queima do mesmo.

Comparando os intervalos de operação para o motor DC desta seção e do *capítulo 3*, observa-se que o melhor desempenho para a entrada de controle é obtido para $\Delta t_{op} = 5\text{seg}$. Isso porque, quando o motor está operando com $\Delta t_{op} = 50\text{seg}$, a entrada de controle apresenta valores de tensão muito baixos, tornando muito lenta a resposta do sistema. Por outro lado, quando $\Delta t_{op} = 1\text{seg}$, a entrada de controle requerida, para uma resposta considerada razoável, apresenta valores de tensão muito elevados para serem aplicados na prática.

Analisando os casos apresentados em função da constante de tempo dominante do sistema, que denotaremos por T_d , pode-se chegar à seguinte conclusão, tomada como básica no decorrer do

restante desta pesquisa:

Conjectura 4.1 : O resultado do método do gradiente conjugado só é satisfatório quando ΔT_{op} for, aproximadamente, um valor entre T_d e $3T_d$.

Com base na Conjectura 4.1, pode-se deduzir que é inviável a idéia de se utilizar a dilatação do intervalo ΔT_{op} para se efetuar o controle ótimo em tempo real pelo método do gradiente conjugado.

4.5. Extensão do Método do Gradiente Conjugado com o Tempo Final Livre

Em função da Conjectura 4.1 apresentada, o método do gradiente conjugado deve operar em um intervalo de tempo limitado, com comprimento máximo de $3T_d$, onde T_d é a constante de tempo dominante do sistema. No caso do motor DC, por exemplo, $T_d = 3\text{seg}$ e o tempo final t_f escolhido foi 5 segundos, dentro, portanto, do intervalo ideal. Ainda, como o sistema é operado no intervalo $[0,5]$ segundos, $\Delta T_{op} = 5\text{seg}$.

Considerando que o intervalo de operação, ΔT_{op} , deve ficar limitado a essa região de maior eficiência, o seguinte algoritmo pode ser empregado para resolver o problema do controle

ótimo com tempo final livre.

ALGORITMO PARA O CONTROLE ÓTIMO COM TEMPO FINAL LIVRE

- (1) Considerando que o intervalo global de operação do sistema é dado por $[t_0, t]$, onde t é livre, e que ΔT_{op} é o intervalo ideal para operação do método do gradiente conjugado, calcula-se, inicialmente:

$$t_i = t_0 ;$$

$$t_f = t_0 + \Delta T_{op} ;$$

$$\mathbf{x}_0 = \mathbf{x}(t_0) .$$

- (2) Enquanto $t \geq t_f$ executa-se:

- (3.1) Calcula-se o controle ótimo no intervalo $[t_i, t_f]$, pelo método do gradiente conjugado convencional, com \mathbf{x}_0 sendo a condição inicial sobre as variáveis de entrada e $\mathbf{u}_0(t)$, a estimativa inicial do controle;

- (3.2) Calcula-se:

$$t_i = t_f ;$$

$$t_f = t_f + \Delta T_{op} ;$$

$$\mathbf{x}_0 = \mathbf{x}(t_i) .$$

4.5.1. Análise dos Resultados

O algoritmo para controle ótimo com tempo real livre, apresentado anteriormente, foi implementado, e, para sua análise, foi usado o caso do motor DC (*seção 3.3.2*) com os parâmetros considerados na *seção 4.4*, a saber: $w_1 = 10000$; $R = 5$; $u_0 = 0$; 5 iterações e 1 reinicialização do método; e $\Delta T_{op} = 5\text{seg}$. O intervalo global de operação do sistema considerado foi de 50 seg, implicando em 10 execuções do método do gradiente conjugado no algoritmo analisado (10 intervalos ΔT_{op}).

Os resultados obtidos estão ilustrados nas *figuras 4.11* e *4.12*, onde podem ser comparados com o melhor caso obtido com o método de dilatação do intervalo de operação, da *seção 4.4*. Observando as curvas de resposta do sistema para ambos os casos, pode-se observar, na *figura 4.11*, que o algoritmo para controle ótimo com tempo final livre apresenta um resultado qualitativamente melhor, tendo atingido praticamente os 10 radianos pouco antes dos 5 segundos.

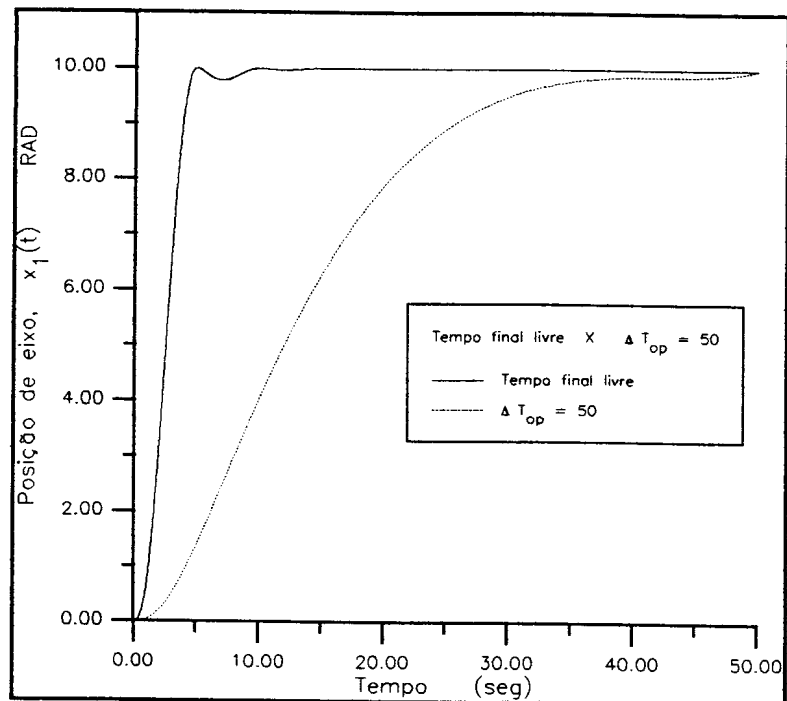


Figura 4.11 - Posição do eixo: dilatação do intervalo de operação x tempo final livre.

Após os 5 segundos, a curva apresenta uma pequena sobre-elevação, visto que, em $t = 5\text{seg}$ (tempo final para o primeiro intervalo $\Delta T_{op}(1)$), a posição é 9,995 radianos, a velocidade é 0,041 radianos/segundo e a corrente de campo é -0,093 amperes. Com esses valores, no início do próximo intervalo, $\Delta T_{op}(2)$, o motor gira no sentido de se afastar da posição desejada de 10 radianos, indo para valores menores.

Na primeira parte do segundo intervalo, $\Delta T_{op}(2)$, observa-se, então, que a tensão de controle (entrada) é positiva, tentando reverter a rotação do eixo e levá-lo de volta aos 10 radianos, como pode ser visto na figura 4.12.

Como era de se esperar, a potência aplicada no motor é muito maior no presente

caso, principalmente no primeiro intervalo, $\Delta T_{op}(1)$, quando o eixo se desloca de 0 a 10 radianos, praticamente.

No algoritmo para controle ótimo com tempo final livre, a cada intervalo ΔT_{op} , executa-se o método do gradiente conjugado,

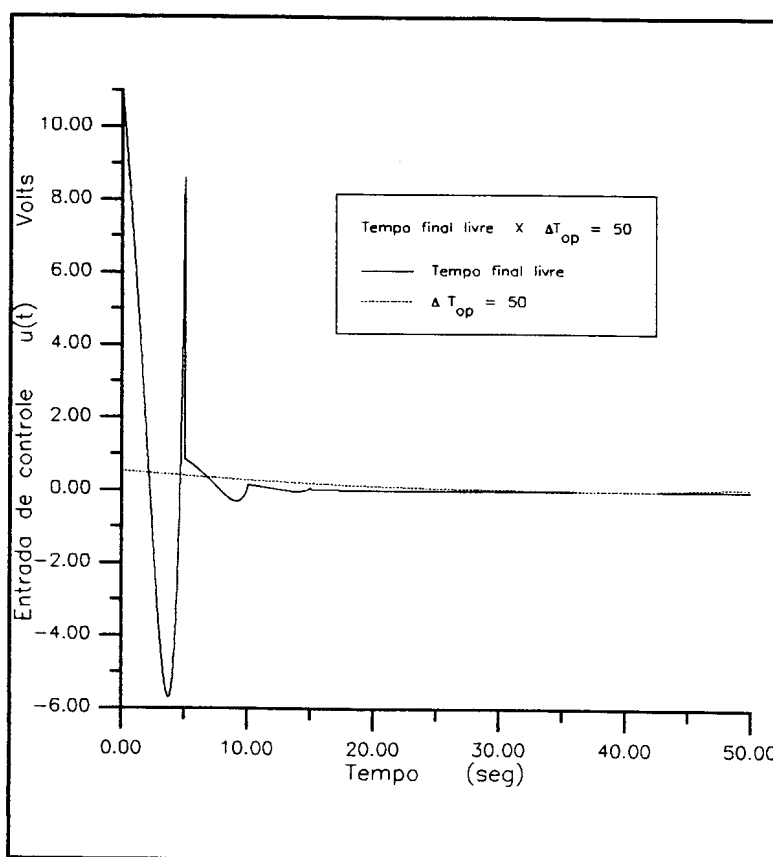


Figura 4.12 - Entradas de controle: dilatação do intervalo de operação x tempo final livre.

para obter o controle ótimo neste intervalo. Uma vez que o instante inicial para o intervalo $\Delta T_{op}(k)$ é o instante final para o intervalo $\Delta T_{op}(k-1)$, o valor final do estado $\mathbf{x}(t)$ em $\Delta T_{op}(k-1)$ também é tomado como valor inicial $\mathbf{x}_0(t)$ em $\Delta T_{op}(k)$. Entretanto, no algoritmo proposto, para cada intervalo ΔT_{op} , a estimativa inicial da entrada de controle, $u_0(t)$, utilizada no primeiro intervalo $\Delta T_{op}(1)$, repete-se em todos os outros intervalos ΔT_{op} .

Uma outra possibilidade seria escolher o valor final de $u(t)$ em um intervalo $\Delta T_{op}(k-1)$ como estimativa inicial para o controle, $u_0(t)$, para o intervalo $\Delta T_{op}(k)$. Esta possibilidade foi investigada, modificando-se o algoritmo proposto. O efeito sobre a resposta do sistema é mostrado na figura 4.13, enquanto as entradas de controle resultantes estão na figura 4.14.

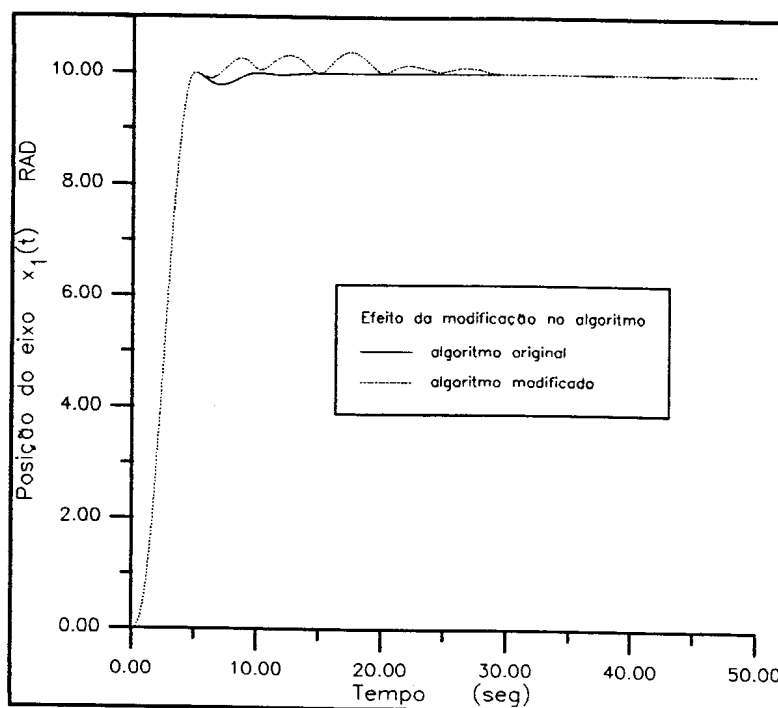


Figura 4.13 - Efeito dos valores iniciais de $u_0(t)$ sobre a resposta do sistema.

Da resposta do sistema, na figura 4.13, pode-se observar que o algoritmo modificado leva a uma oscilação maior na resposta que o original. Ainda, o sistema entra em regime permanente após os 30 segundos, ao passo que, no algoritmo original, com aproximadamente 12 segundos o sistema já está em regime permanente.

A figura 4.14 ilustra bem o instante em que o sistema entra em regime para os dois casos, quando então as entradas de controle caem para praticamente zero.

Esperava-se que a resposta no caso modificado fosse melhor que a do original, pois, próximo do regime, a entrada de controle para o intervalo $\Delta T_{op}(k)$ mantém certa semelhança com a entrada para $\Delta T_{op}(k-1)$.

Uma possível explicação para esse comportamento é: se o método parte de uma região mais afastada do mínimo então ele vai apresentar uma razão de convergência maior que se partisse de uma região muito próxima.

No algoritmo original, o fato de, no intervalo $\Delta T_{op}(k)$, resgatar-se o mesmo valor inicial de $u_0(t)$ do intervalo $\Delta T_{op}(k-1)$, é semelhante à sugestão de Fletcher e Reeves [FLETCHER, 64], pela qual se reinicializa o método do gradiente conjugado a cada $(n+1)$ iterações, onde n é a dimensão do funcional cujo mínimo é buscado.

Considerando os resultados analisados, o algoritmo para controle ótimo com tempo final livre original é o mais eficiente,

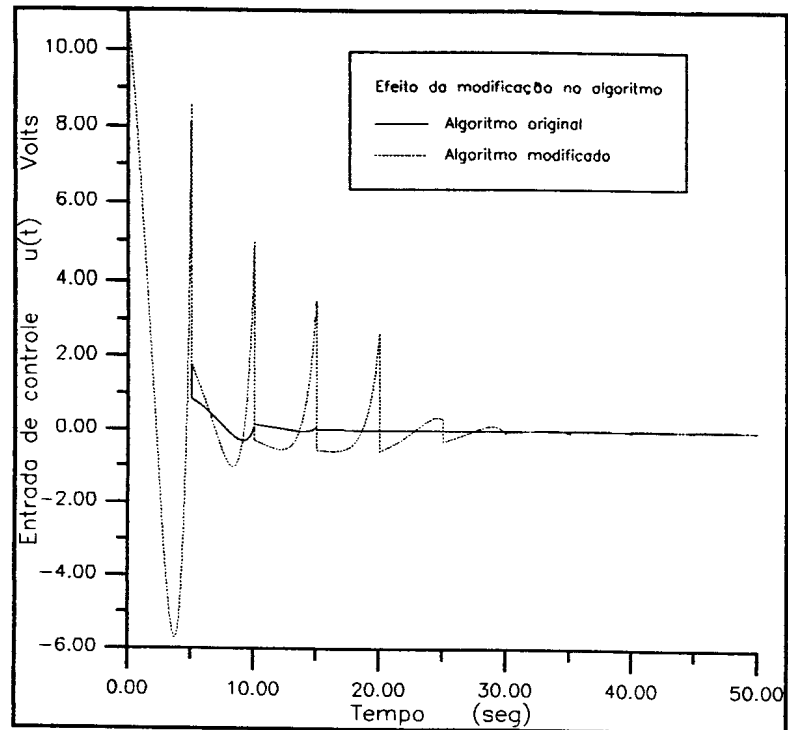


Figura 4.14 - Efeito dos valores iniciais de $u_0(t)$ sobre as entradas do sistema.

sendo, portanto, escolhido para a abordagem do problema do controle ótimo em tempo real da próxima seção.

4.6. Controle Ótimo em Tempo Real de Processos via Método com Tempo Final Livre

O algoritmo para o controle ótimo com tempo final livre, descrito na seção anterior, proporciona um modo de aplicação do método do gradiente conjugado para o controle ótimo, sem a condição do tempo final de operação ser fixado, de forma que este controle pode ser calculado durante todo o tempo de operação do processo.

Usando o algoritmo proposto, diferentes implementações de sistemas de controle ótimo em tempo real podem ser realizadas, dependendo da ocorrência de realimentação das variáveis do processo para o modelo computacional e do intervalo de tempo em que o controle deve ser calculado. Tais implementações serão propostas e analisadas nas seções subseqüentes.

Na aplicação do algoritmo para controle ótimo com tempo final livre no controle ótimo em tempo real, alguns parâmetros do tempo são relevantes para a caracterização dos sistemas.

O primeiro parâmetro é o tempo t , durante o qual o sistema estará em operação. Esse tempo deve ser múltiplo de ΔT_{op} , de forma que $t = N + \Delta T_{op}$, onde N é um inteiro maior ou igual a 1.

O segundo parâmetro é o intervalo de operação em que o método do gradiente conjugado será calculado, ΔT_{op} , que, com base na Conjectura 4.1 (seção 4.4.2), deve valer de 1 a 3 T_d , onde T_d é a constante de tempo dominante do sistema.

O terceiro parâmetro é o intervalo de discretização dt , utilizado como passo de integração nos algoritmos de Runge-Kutta e no método de Euler para integração de funções contínuas. Esse intervalo é importante porque, além de definir a precisão com que as integrações são realizadas, ele define a quantidade de memória necessária para o armazenamento das variáveis do sistema. O intervalo de discretização define, ainda, o tempo que um sinal deve permanecer nos conversores Digital/Análogo (D/A), bem como o intervalo de amostragem dos conversores Análogo/Digital (A/D).

O quarto parâmetro é o tempo de execução t_{ex} que um determinado computador consome para calcular as entradas de controle no intervalo ΔT_{op} , pelo método do gradiente conjugado.

Outros dois parâmetros adicionais, relacionados com a entrada e saída de dados no computador, podem ser definidos. Um deles, o tempo para a entrada de dados no computador t_e , é definido como o tempo que o dado leva para ir do processo até a memória do computador que está executando o algoritmo do gradiente conjugado. Essa definição abrange o caso de serem utilizados dois computadores: um para fazer a interface com o processo e outro para executar o algoritmo de controle. Nesse tempo devem ser computados, inclusive, o tempo gasto pelos conversores A/D's, o tempo gasto na comunicação entre os processadores e o tempo de acesso à memória

de ambos os processadores. De forma análoga, o outro parâmetro adicional, o tempo t_s , pode ser definido como o tempo que o dado leva para ir da memória do computador que está executando o algoritmo até o processo.

Normalmente, t_e e t_s podem ser desprezados quando ocorrerem as seguintes situações:

$$(a) \quad t_e + t_s \ll t_{ex} < dt;$$

$$(b) \quad t_e + t_s \ll dt < t_{ex} < \Delta T_{op}.$$

4.6.1. Controle Ótimo em Tempo Real em Malha Aberta

O algoritmo para controle ótimo com tempo final livre gera as entradas de controle $u(t)$, que irão minimizar o funcional de custo durante todo o tempo t de operação do sistema. Estas entradas podem ser aplicadas diretamente no processo para implementar o controle ótimo em tempo real, segundo o esquema da figura 4.15.

Nesta forma, o controle ótimo é aplicado no processo em malha aberta, ou seja, as entradas são aplicadas segundo a lei de controle calculada sobre o modelo, quer as variáveis do modelo sigam ou não as do processo.

Da figura 4.15 pode-se observar que a condição para que as variáveis de estado do modelo computacional, $x_m(t)$, sigam as variáveis do processo, $x_p(t)$, é que as equações diferenciais do

modelo computacional descrevam exatamente o modelo físico e que as perturbações externas aplicadas ao processo sejam nulas.

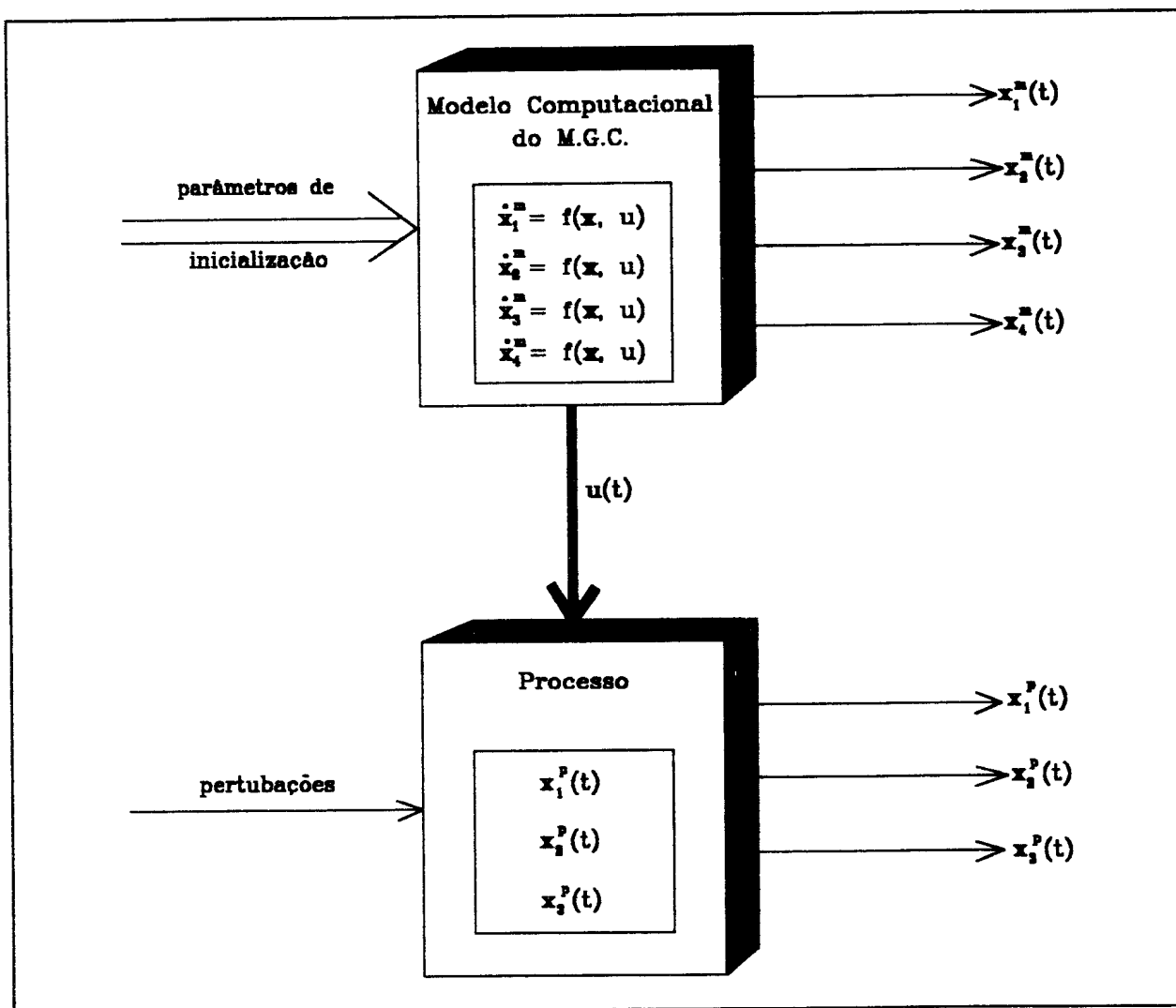


Figura 4.15 - Esquema para o controle ótimo em tempo real em malha aberta.

Embora o método do gradiente conjugado permita que se trabalhe com equações diferenciais não lineares, nem sempre é possível encontrar uma equação diferencial que descreva exatamente o processo físico. Mesmo que fosse encontrado um modelo que assim o fizesse, seria impossível prever todas as perturbações externas

aplicadas ao sistema durante sua operação. Entretanto, desde que o sistema tenha equações do modelo que o descrevam razoavelmente e não tenha o intervalo de operação demasiadamente longo e perturbações externas de grande magnitude, então as variáveis de estado do modelo podem seguir as variáveis do processo com uma margem de erro tolerável, podendo ser aplicado o controle ótimo em malha aberta.

A aplicação do controle ótimo em tempo real em malha aberta, por sua vez, pode ser feita de duas maneiras: com restrições severas no tempo de processamento e com restrições brandas no tempo de processamento, conforme colocado a seguir.

Controle Ótimo em Tempo Real com Restrições Severas no Tempo de Processamento para Sistemas de Malha aberta

Considerando um sistema que deva ser operado no intervalo de tempo total de $[t_0, t]$, onde $t = (t_0 + N * \Delta T_{op})$, o controle ótimo, neste caso, deve ser calculado para cada um dos N intervalos de operação do sistema (ΔT_{op}) em um tempo t_{ex} menor que dt . Assim, quando o sistema começa a operar em t_0 , a entrada de controle ótima $u(t_0+dt)$ está disponível para o processo em $(t_0 + dt)$. Um diagrama de tempo para esta situação está ilustrado na figura 4.16.

Nesta abordagem, o processador deve ser suficientemente rápido para efetuar todo o algoritmo de controle ótimo pelo método do gradiente conjugado para o intervalo ΔT_{op} , no tempo t_{ex} . Se o

sistema trabalha com m intervalos de discretização em ΔT_{op} , então o processador vai ficar ocioso por um período de $(m-1)*dt$, tempo este que pode ser aproveitado pelo processador apenas para enviar dados ao processo.

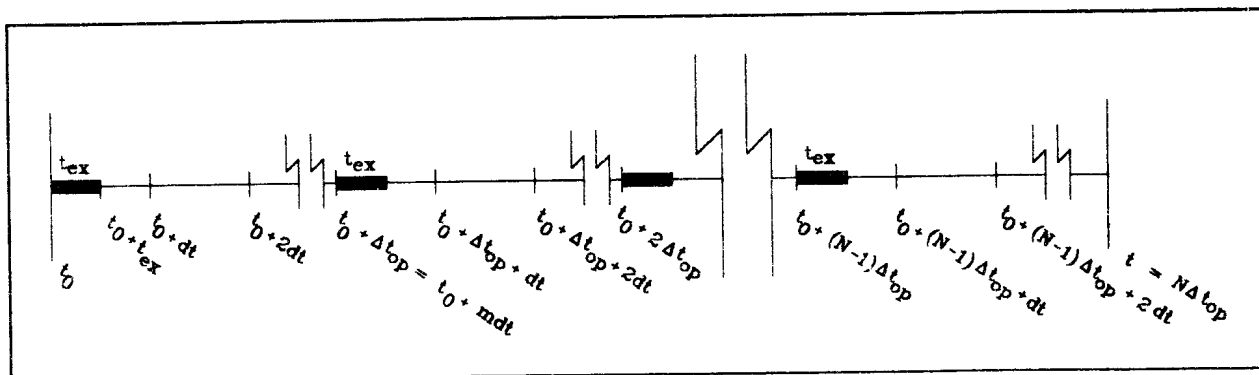


Figura 4.16 - Diagrama de tempo para o controle ótimo em tempo real com restrições severas de tempo em sistemas de malha aberta.

Controle Ótimo em Tempo Real com Restrições Brandas no Tempo de Processamento em Sistema de Malha Aberta

Em sistemas de malha aberta, o caso anterior implica em desperdício da capacidade do processador, pois o mesmo fica ocioso por um período de tempo muito grande. Uma forma alternativa seria utilizar um processador com velocidade suficiente para calcular as entradas de controle para o intervalo ΔT_{op} . Para esse intervalo, a condição sobre o tempo de execução do algoritmo do gradiente conjugado é $t_{ex} < \Delta T_{op}$. O diagrama de tempo para este caso está ilustrado na figura 4.17.

No exemplo do motor DC, onde foi usado 150 intervalos

de discretização em ΔT_{op} , o controle com restrições brandas permite que seja utilizado um processador, pelo menos, 150 vezes mais lento.

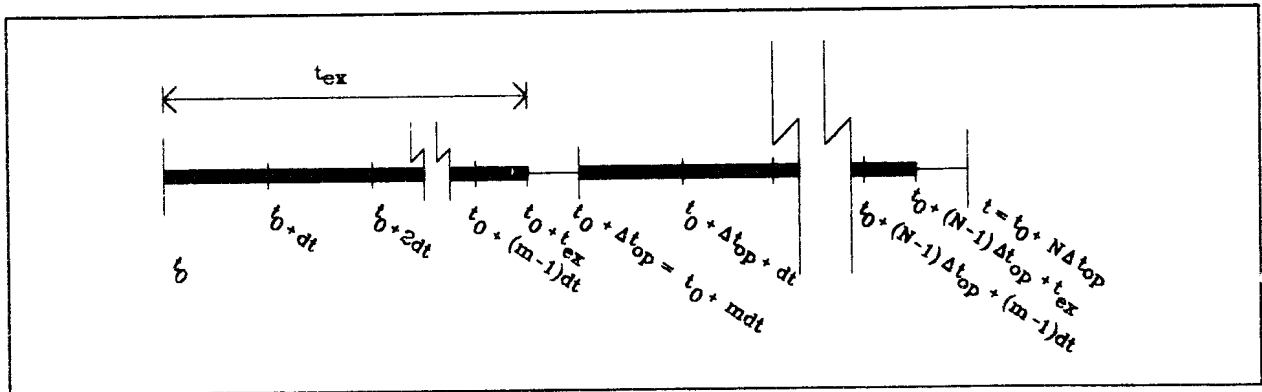


Figura 4.17 - Diagrama de tempo para o controle ótimo em tempo real com restrições brandas de tempo em sistemas de malha aberta.

Considerando que as entradas de controle ótimas serão enviadas ao processo sem realimentação (malha aberta), do ponto de vista do processo, não há diferença entre as duas formas de cálculo. Sendo assim, o controle com restrições brandas é o preferido, pois tal implementação pode ser realizada com um processador muito mais barato.

4.6.2. Controle Ótimo em Tempo Real em Malha Fechada

O controle ótimo aplicado ao processo em malha aberta, como esquematizado na figura 4.15, pode levar a uma disparidade entre as variáveis de estado do modelo x_m e as variáveis do processo x_p , em um instante t qualquer.

As causas para essa discordância entre as variáveis do modelo e do processo podem ser devido a perturbações externas não previstas no processo, erro de modelamento, modelo inadequado para descrever o processo em certas condições de operação, erros de truncamento e arredondamento no processo numérico, bem como devido à precisão dos conversores D/A's.

Essas discrepâncias podem ser eliminadas, comparando-se as variáveis reais do processo com as variáveis do modelo, e, no caso de algum erro, modificando-se as entradas de controle no sentido de corrigir tal erro. Nesse caso, as entradas de controle ótimas são aplicadas no processo utilizando um esquema de realimentação, ou seja, em malha fechada. Uma ação desse tipo está esquematizada na figura 4.18.

A aplicação do controle ótimo em tempo real em malha fechada também pode ser feita de duas maneiras: com restrições severas e com restrições brandas no tempo de processamento, como será visto a seguir.

Controle Ótimo em Tempo Real com Restrições Severas no Tempo de Processamento para Sistemas de Malha Fechada

A implementação da realimentação pode ser realizada de várias maneiras, ressaltando-se, porém, que as novas entradas de controle obtidas com realimentação ainda devem minimizar o funcional de custo.

O esquema de realimentação proposto a seguir é uma adaptação do algoritmo de controle ótimo com tempo final livre, para o caso de restrições severas no tempo de processamento.

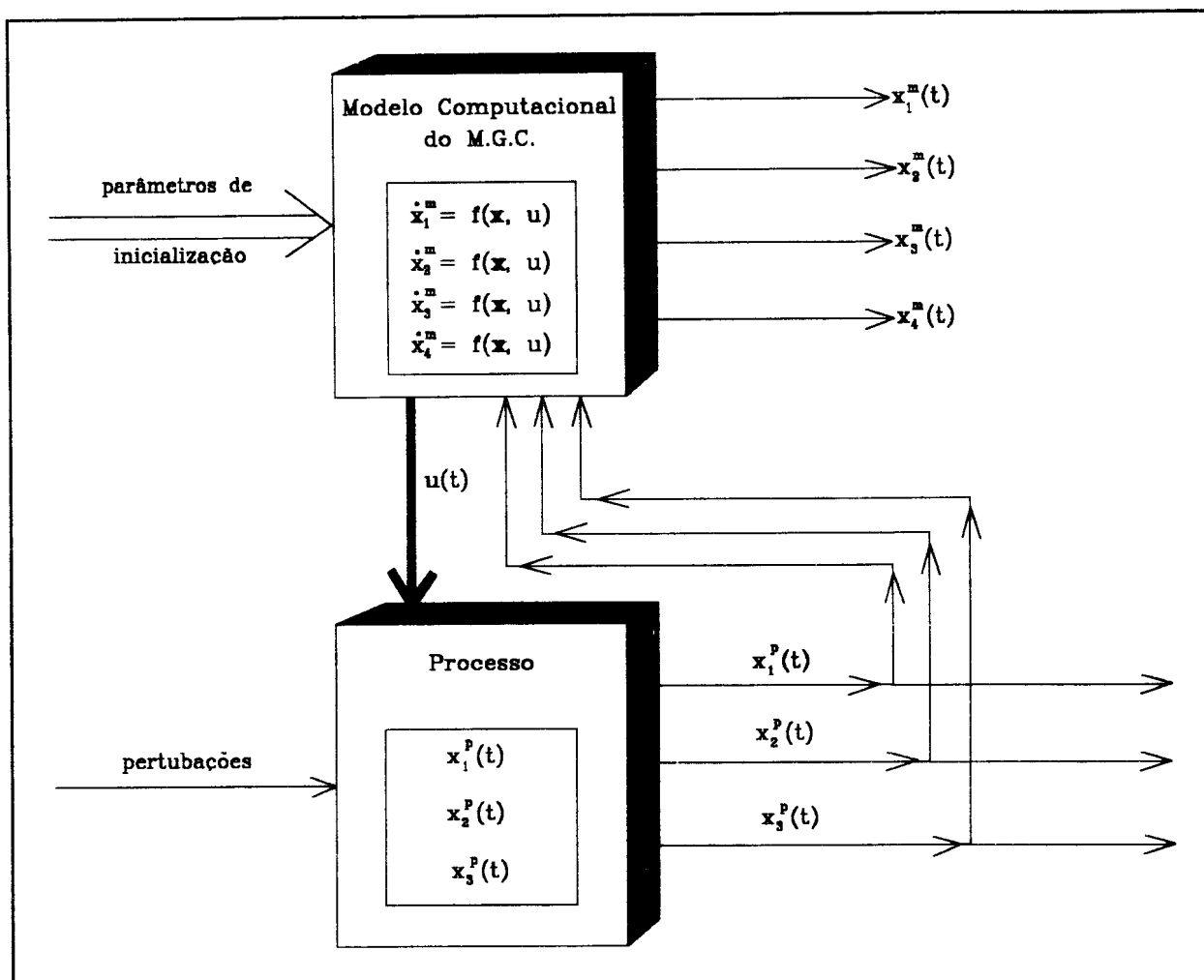


Figura 4.18 - Esquema para o controle ótimo em tempo real em malha fechada.

Sob as considerações desta seção, o algoritmo para o controle ótimo com tempo final livre, da *seção 4.5*, aplica-se, obviamente, às equações do modelo e não às variáveis do processo real. Desse modo, em função da *figura 4.18*, o passo (3.2) daquele algoritmo é:

(3.2) Calcula-se:

$$t_i = t_f ;$$

$$t_f = t_f + \Delta T_{op} ;$$

$$\mathbf{x}_0^m = \mathbf{x}^m(t_i) .$$

A modificação proposta é substituir $\mathbf{x}_m(t_i)$ por $\mathbf{x}_p(t_i)$. Assim, a cada intervalo ΔT_{op} em que o método do gradiente conjugado será calculado, estabelece-se, como condição inicial para o estado do modelo, os valores reais das variáveis do processo. No primeiro intervalo, entretanto, a condição inicial sobre o estado do modelo é seu próprio valor em t_0 , dado pelas condições iniciais do problema, ou seja, $\mathbf{x}_0 = \mathbf{x}(t_0)$.

Assim, o passo (3.2) passa a ser:

(3.2)' Calcula-se:

$$t_i = t_f ;$$

$$t_f = t_f + \Delta T_{op} ;$$

$$\mathbf{x}_0^m = \mathbf{x}^p(t_i) .$$

Em termos de diagrama de tempo, continua valendo o diagrama da figura 4.16.

As restrições sobre o tempo de processamento aparecem porque, para cada intervalo ΔT_{op} , o total do tempo de execução t_{ex} do método do gradiente conjugado, mais o tempo para leitura das variáveis do processo t_s , mais o tempo de saída da variável de controle t_g , deve ser menor que dt .

Aplicação no caso do motor DC

Para testar o algoritmo modificado, foi escolhido o caso do motor DC, para o qual fez-se uma simulação das variáveis do motor, estando este sujeito a perturbações externas. A simulação foi realizada para cada instante $t = \Delta T_{op}(k)$, onde $1 < k \leq N$, considerando para as variáveis do processo:

$$x_i^p = x_i^m * (1 + 0,1 * R) \quad \text{para } i=1,2,3 \quad (4.6-1)$$

onde R é um número pseudo-aleatório gerado no computador.

A equação (4.6-1) faz com que as variáveis do processo contenham um erro de $\pm 10\%$ em relação às variáveis do modelo, significando um erro absoluto de até 20%.

Em aplicações práticas, o erro é bem menor que o valor simulado, geralmente não ultrapassando 5% em termos absolutos. Entretanto, o valor de 10% foi escolhido para verificar a estabilidade do sistema. Além disso, essa perturbação é atribuída a todas as variáveis de estado (em casos práticos, algumas variáveis

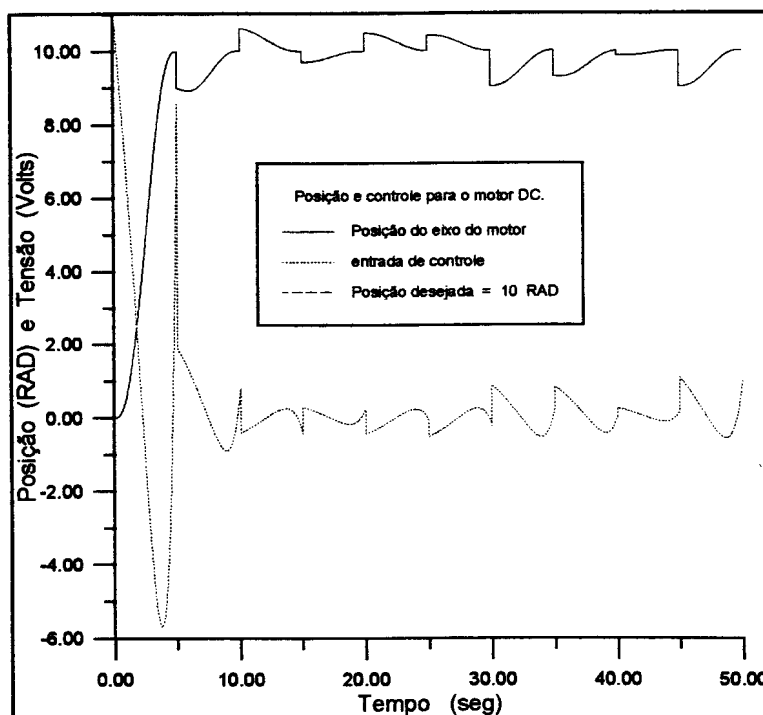


Figura 4.19 - Tempo real, malha fechada, restrições severas de tempo: erro de $\pm 10\%$ nas variáveis do processo.

podem não estar sujeitas a perturbações).

O resultado da simulação é mostrado na figura 4.19, onde pode-se observar que a posição final para cada intervalo de operação do método do gradiente conjugado, ΔT_{op} , está bastante próxima dos 10 radianos. Observa-se, ainda, que, no início dos intervalos ΔT_{op} , a posição está afastada dos 10 radianos, o que se deve ao erro introduzido nas variáveis do processo. É interessante ressaltar que, nesses intervalos, a entrada de controle atua no sentido de diminuir o erro de posição.

Controle Ótimo em Tempo Real com Restrições Brandas no Tempo de Processamento para Sistemas de Malha Fechada

Embora os resultados obtidos no item anterior (com restrições severas no tempo de processamento), sejam qualitativamente muito bons do ponto de vista de controle de processos, as exigências quanto ao tempo de processamento podem inviabilizar muitas aplicações práticas.

Outra desvantagem do método anterior, que pode ser observada na figura 4.16, é que o processador fica sobrecarregado em um intervalo de tempo dt e ocioso em outro intervalo muito maior, $(m-1)*dt$.

Tentando suavizar as restrições quanto ao tempo de processamento e evitar que o processador fique ocioso por um período muito grande do tempo, propõe-se uma nova alteração do algoritmo de controle ótimo com tempo final livre (seção 4.5). Tal

alteração permitirá a solução de problemas de controle ótimo em tempo real com restrições brandas no tempo de execução (*soft real time*) e com realimentação das variáveis do processo.

No caso do controle em tempo real com restrições severas no tempo de processamento (*hard real time*), visto anteriormente, para um dado intervalo $\Delta T_{op}(k)$ é feita a leitura das variáveis do processo e, para todo o intervalo $\Delta T_{op}(k)$, o controle ótimo é calculado pelo método do gradiente. Esse cálculo do controle ótimo deve ser efetuado antes da decorrência do próximo intervalo dt , quando o primeiro valor da entrada de controle já deve ser passado para o processo. A partir desse ponto, o processador será utilizado somente para passar os $(m-1)$ valores restantes.

Os seguintes aspectos básicos foram considerados no desenvolvimento da modificação do algoritmo:

- . quando o sistema estiver em regime permanente, os valores das variáveis de estado em $\Delta T_{op}(k-1)$ não vão distoar muito dos valores em $\Delta T_{op}(k)$;

- . se o intervalo de operação do gradiente conjugado, ΔT_{op} , foi escolhido segundo a Conjectura 4.1, então após o primeiro intervalo, $\Delta T_{op}(1)$, o sistema já estará em regime permanente;

- . como as entradas de controle são calculadas para todo o intervalo ΔT_{op} , somente após a passagem deste tempo é que o sistema vai necessitar de outras entradas. Então, é interessante

que o intervalo ΔT_{op} seja aproveitado pelo processador para calcular as próximas entradas.

Tendo em vista os pontos básicos apresentados, as seguintes modificações podem ser efetuadas no passo (3.2) do algoritmo de controle ótimo com tempo final livre, da *seção 4.5*, levando-se em conta que, durante toda a execução do algoritmo, serão considerados os intervalos $\Delta T_{op}(k)$, onde $k = 1, \dots, N$:

(3.2)'' Calcula-se:

$$t_i = t_f ;$$

$$t_f = t_f + \Delta T_{op} ;$$

$$\text{se } k \leq 2$$

$$\text{então } \mathbf{x}_0^m = \mathbf{x}^m(t_i)$$

$$\text{senão } \mathbf{x}_0^m = \mathbf{x}^p(t_i - \Delta T_{op}) .$$

O algoritmo definido na *seção 4.5*, com o passo (3.2) substituído pelo (3.2)'' acima, constitui uma seqüência de cálculo em linha de montagem ou canalização (*pipeline*).

Inicialmente, as entradas de controle são calculadas pelo método do gradiente conjugado, para o intervalo $\Delta T_{op}(1)$, com $\mathbf{x}_0^m = \mathbf{x}(t_0)$ dado pelas condições iniciais.

Dando seqüência, o processo é inicializado e as entradas começam a ser passadas para o processo, uma a cada intervalo dt . Simultaneamente, o processador inicia o cálculo das entradas de controle para o próximo intervalo, $\Delta T_{op}(2)$, onde as condições

iniciais para as equações de estado são os valores das variáveis de estado no instante final do intervalo $\Delta T_{op}(1)$, já calculado.

No terceiro intervalo, $\Delta T_{op}(3)$, as condições iniciais para a equação de estado são os valores das variáveis do processo medidos no final do intervalo $\Delta T_{op}(1)$.

Portanto, no intervalo $\Delta T_{op}(k)$, para $2 < k \leq N$, as condições iniciais para a equação de estado são os valores das variáveis do processo medidos no final do intervalo $\Delta T_{op}(k-2)$.

Esta abordagem define um processo de canalização (*pipeline*) de dois estágios, onde, enquanto as entradas para um intervalo estão sendo calculadas, as entradas já calculadas no intervalo anterior estão sendo enviadas ao processo. A vantagem deste esquema é que, se o intervalo ΔT_{op} foi discretizado em m subintervalos, o processador pode dispor de um intervalo de tempo de até $(m-1) \cdot dt$ para calcular as entradas de controle. A carta de tempo da figura 4.17 ilustra essa situação.

Considerando que o tempo para entrada de dados do processo até a memória do computador, t_e , e o tempo de saída de dados da memória até o processo, t_s , são muito pequenos, comparados com o tempo de cálculo das entradas de controle, então um único processador pode ser usado para executar os dois estágios da canalização. Quando as operações de entrada e saída para o processo tomarem muito tempo do processador, pode ser utilizado um processador dedicado a essa tarefa.

Aplicação no caso do motor DC

Para efeito de comparação, o algoritmo modificado para cálculo em linha de montagem foi programado e aplicado ao caso do motor DC.

Inicialmente, o programa foi executado considerando não haver erro entre as variáveis do processo e as variáveis do sistema. Os resultados obtidos podem ser vistos na figura 4.20.

A curva pontilhada representa o caso de tempo real com restrições severas de tempo e com realimentação nas variáveis do processo, porém desconsiderando a ocorrência de erro entre as variáveis do processo e as do modelo. Sob tais condições, a curva coincide com o caso de controle ótimo com tempo final livre (figura 4.11).

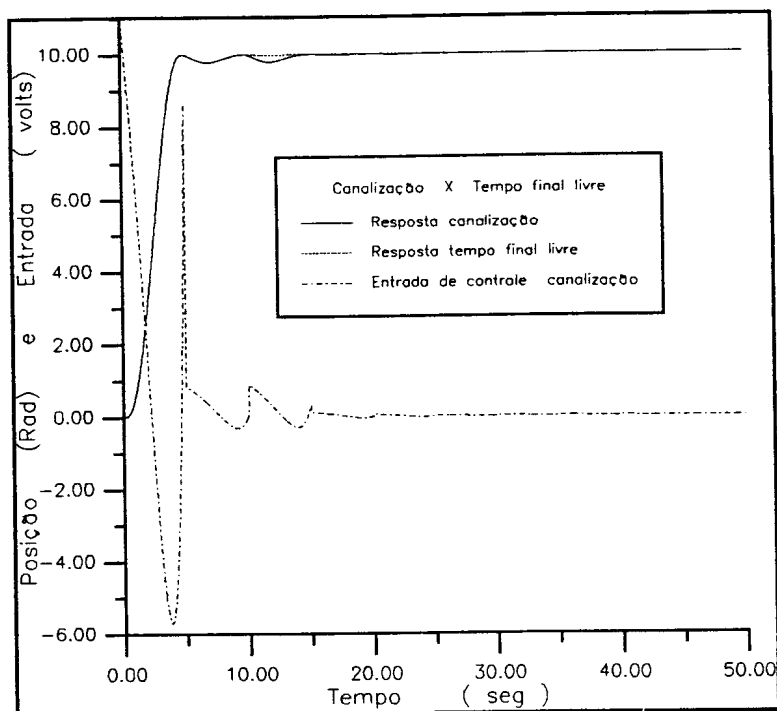


Figura 4.20 - Caso de canalização sem erro nas variáveis do processo x tempo final livre.

Na figura 4.20 pode-se observar que, nesse caso, o sistema requer mais dois intervalos ΔT_{op} para se estabilizar, no entanto, a curva não difere muito do caso ideal (linha pontilhada). Esta oscilação extra se deve à repetição das entradas de controle

em $\Delta T_{op}(2)$ e $\Delta T_{op}(3)$, quando os estágios da canalização não estão completos (linha tracejada na figura 4.20).

Considerando que os resultados obtidos foram bastante satisfatórios, o próximo passo foi simular um sistema de malha fechada com um erro máximo de $\pm 10\%$ nas variáveis do processo, em relação às variáveis do modelo. O esquema para o controle ótimo com realimentação pode ser visto na figura 4.18.

A simulação obedeceu a equação (4.6.2-1), com os números pseudo-aleatórios gerados com os mesmos parâmetros de inicialização do caso com restrições severas de tempo (sem canalização). Dessa forma, os números pseudo-aleatórios se repetem e os dois casos podem ser comparados.

O resultado para a posição do eixo pode ser observado na figura 4.21, enquanto que a entrada de controle, na figura 4.22. Nestas figuras, as curvas pontilhadas representam os casos com restrições severas no tempo de processamento (caso sem *pipeline*).

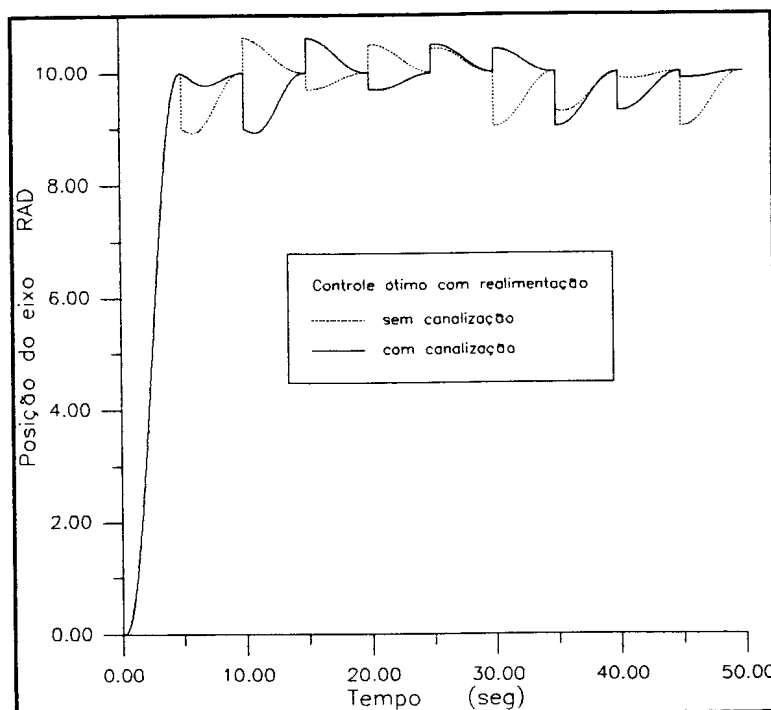


Figura 4.21 - Posição do eixo: com canalização e sem canalização.

Na figura

4.21 observa-se que, a cada intervalo de operação do gradiente conjugado, as posições do eixo coincidem no tempo final. Isto significa que ambos os casos acabam levando o eixo à posição desejada, a despeito das perturbações nas variáveis do processo.

Pode-se observar, ainda, que nos dois primeiros intervalos de operação do gradiente conjugado, ΔT_{op} , é a curva para o caso que usa canalização que se aproxima mais da resposta ideal. Isto porque, enquanto os dois estágios não são completados, o caso de canalização usa as variáveis de estado do modelo, que não são afetadas pelo erro.

A figura 4.22 mostra que, em ambos os casos, as variáveis de entrada atuam no sentido de levar o eixo para a posição desejada, com igual eficiência.

Considerando que no caso de restrições brandas (canalização) o tempo de processamento é aproximadamente $(m-1)$ vezes menor,

este caso é realmente o mais indicado para implementação, uma vez que as restrições sobre o desempenho do processador são muito menos severas. Ainda, o processador é usado com mais eficiência, pois

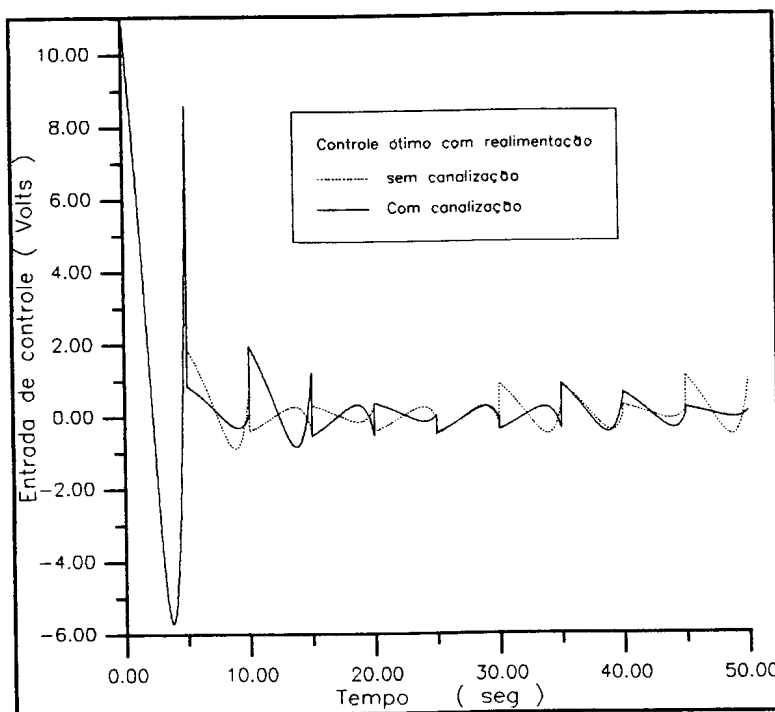


Figura 4.22 - Entrada de controle: com canalização e sem canalização.

praticamente não há tempo ocioso nesse processo. Além disso, como mostrado neste exemplo, os resultados, para ambos os casos, são qualitativamente semelhantes.

No próximo capítulo serão analisados os requisitos de desempenho para os processadores usados na implementação do controle ótimo em tempo real.



CAPÍTULO 5

Análise de Desempenho de Processadores para o Controle Ótimo em Tempo Real

5.1. Introdução

A solução de problemas de controle ótimo em tempo real implica no tempo de execução t_{ex} estar sujeito a uma restrição, seja ela "severa" (*hard*) ou "branda" (*soft*). O tempo de execução t_{ex} , como definido no capítulo anterior, é o tempo que o processador leva para calcular as entradas de controle para o intervalo de operação ΔT_{op} , para o qual o método do gradiente será aplicado.

No caso de controle ótimo com restrições severas de tempo, t_{ex} deve ser menor que dt , onde dt é o intervalo de discretização ou passo de integração no algoritmo de Runge-Kutta. Quando as restrições são brandas, t_{ex} apenas deve ser menor do que ΔT_{op} , uma situação mais fácil de se implementar.

Em ambos os casos, o processador utilizado para os cálculos deve ter velocidade suficiente para que t_{ex} possa satisfazer as restrições acima. Embora a velocidade seja uma condição necessária para que um processador possa ser utilizado na implementação do controle ótimo em tempo real, ele deve possuir outras características adicionais, como dimensões físicas reduzidas e baixo custo, bem como possuir características de um sistema aberto. Tais características serão discutidas nos próximos parágrafos.

As dimensões físicas do processador constituem um fator relevante. Por exemplo, se for necessário empregar um supercomputador, que ocupe toda uma sala, para satisfazer as restrições de velocidade sobre t_{ex} , dificilmente ele poderá ser acoplado ao processo. Na maioria das aplicações de controle ótimo em tempo real, o processador deve ser incorporado ao processo ou então ser embarcado, como no caso das aplicações automobilísticas, náuticas e aeronáuticas. Logo, as dimensões físicas do processador devem ser compatíveis com tais aplicações.

Outro fator que é determinante para a escolha do processador é o custo. Não é viável utilizar um processador para o controle de um dado processo quando este processador custar muito mais caro que o próprio processo. Normalmente é desejável que o custo do computador seja apenas uma fração do custo total do processo a ser controlado.

Finalmente, o processador deve ser considerado um sistema aberto, tanto a nível de circuitos e interfaces

(*hardware*) quanto a nível de programas (*software*).

Um sistema é considerado aberto quando ele "implementa um número suficiente de especificações abertas para interfaces, serviços e tolera formatos para habilitar os programas de aplicação a serem portáteis, com mudanças mínimas, através de uma ampla gama de sistemas e poderem interoperar com outros aplicativos em sistemas locais e remotos e interagirem com usuários de uma maneira que facilite a portabilidade" [DIGITAL, 91].

Nesta definição, uma especificação aberta é aquela que "é mantida por um processo aberto, de consenso público e consistente com padrões, para acomodar novos desenvolvimentos tecnológicos" [DIGITAL, 91]. Quando estes padrões são ditados por organismos oficiais e reconhecidos pela comunidade de usuários, tem-se um padrão aberto *de juri* ou legal. Por outro lado, quando os padrões são simplesmente aceitos pela comunidade, de uma maneira natural, sem que sejam ditados por organismos oficiais, tem-se um padrão *de facto* ou de fato.

A grande vantagem da utilização de um sistema de computação aberto, tanto a nível de circuitos e interfaces quanto a nível de programas, é que o usuário não fica restrito a um único fabricante. Além disso, o custo tende a ser mais baixo que o de sistemas proprietários; a documentação é obtida facilmente e a um nível maior de detalhes; e existe uma grande disponibilidade de sistemas, interfaces e programas no mercado.

O crescente desenvolvimento dos microcomputadores

baseados em microprocessadores, os torna candidatos naturais à implementação do controle ótimo em tempo real, pois, praticamente, a cada quatro anos surge uma família de microprocessadores mais poderosa, com palavras com o dobro de bits, e a custos equivalentes à anterior ou até mesmo mais reduzidos. Em particular, os microcomputadores tipo "IBM-PC" além de apresentarem as características de velocidade de processamento, porte reduzido e baixo custo, também podem ser considerados como um padrão aberto de fato. Assim sendo, nas próximas seções, o desempenho desta família de microcomputadores será analisado, visando o controle ótimo em tempo real.

5.2. A Linha de Computadores "IBM-PC"

Em 1981, a IBM entrou no mercado de computadores pessoais com o seu modelo "**PC**" (***Personal Computer***). O projeto utilizava o microprocessador Intel 8088, possuía 64 Kbytes de memória RAM, uma única unidade acionadora de discos flexíveis de 5¹/₄ polegadas e custava US\$3.005,00.

Posteriormente, em 1983, a IBM introduziu o **PC-XT** que possui mais três soquetes para expansão do barramento, memória de 256 kbytes (com possibilidade de se chegar até 640 kbytes na placa mãe), disco rígido de 10 Mbytes, duas unidades acionadoras de discos flexíveis de 5¹/₄ polegadas, a um preço de US\$4.995,00. A frequência do relógio ("*clock*") tanto para o PC como para o PC-XT era de 4,77 Mhz [SMARTE, 90].

O microprocessador Intel 8088, em que estes primeiros PCs da IBM estavam baseados, possui 20 linhas para endereços e pode trabalhar internamente com um barramento de dados de 16 bits, porém o barramento externo é de apenas 8 bits. Na época em que este microprocessador foi lançado, esta era uma alternativa interessante, pois ele poderia utilizar diretamente todos os circuitos periféricos projetados para o Intel 8085 - um microprocessador de 8 bits muito utilizado em controle e instrumentação eletrônica. Internamente, o 8088 é idêntico a um microprocessador de 16 bits lançado um ano antes, o Intel 8086. Considerando as 20 linhas de endereço, tanto o 8086 quanto o 8088 podem endereçar diretamente 1 Mbyte de memória.

A limitação do barramento de dados em 8 bits terminou com o lançamento do PC-AT, em outubro de 1984, baseado no microprocessador 80286 da Intel, que é uma unidade de 16 bits completa. A versão inicial foi lançada com uma frequência de relógio de 6 Mhz, logo ampliados para 8, 10 e, posteriormente, para 12,5, 16 e 20 Mhz.

O microprocessador 80286 pode trabalhar com 16 Mbytes de memória real, como consequência de suas 24 linhas de endereço. Além disso, ele pode endereçar 1008 Mbytes de memória virtual, resultando em um total, entre memória real e virtual, de 1 gigabyte. O 80286 pode operar em dois modos: o modo real e o protegido. O modo real reproduz exatamente o modo de operação do 8088, inclusive com sua limitação de até 1 Mbyte no endereçamento de memória. O modo protegido pode usar a capacidade de endereçamento de 16 Mbytes; entretanto, demorou cerca de 3 anos para que

a própria IBM criasse um sistema operacional para tirar proveito do modo protegido: o OS/2.

Um problema que restringiu a expansão do 80286 é a impossibilidade de passar do modo protegido para o modo real sem ter que reinicializar o microprocessador, embora a passagem do modo real para o protegido seja permitida. Outra limitação do 80286 é que ele, a exemplo do 8088, opera com segmentos de memória de 64 kbytes [ROSCH, 90].

Em 1986 foi lançado o AT-386, baseado no Intel 80386, um microprocessador com registradores e barramentos de dados de 32 bits, e de 34 bits para endereços. Permitindo endereçar diretamente 4 gigabytes de memória real e com suporte para gerenciamento de memória virtual de 16 terabytes. Os segmentos de memória neste microprocessador não estão limitados aos 64 kbytes, podendo atingir o tamanho máximo de 4 gigabytes.

De forma a manter a compatibilidade com os membros anteriores da família o 80386 possui um modo de operação real, inclusive sujeito ao limite de 1 megabyte para endereçamento de memória. O microprocessador é sempre inicializado neste modo. É possível passar do modo real para o modo protegido, onde o 80386 passa a funcionar como um 80286, porém com muito mais memória disponível, e com mais flexibilidade para seu gerenciamento. O 80386, diferentemente de seu antecessor, não necessita ser reinicializado para passar do modo protegido para o real.

Existe também um terceiro modo de operação, o modo

virtual, onde é possível simular vários 8088 operando simultaneamente. Neste modo de operação, a memória do 80386 é compartilhada entre todas as máquinas virtuais. O modo virtual facilita a implementação de multitarefa, pois grande parte do trabalho é feito por circuitos e não por programação.

Inicialmente, a frequência do relógio no 80386 era de 12,5 e 16 Mhz, chegando até a 33 Mhz. Há, também, uma versão de 40 Mhz produzida pela AMD, um fabricante de segunda fonte. O 80386 apresenta um recurso extra para aumentar seu desempenho, uma memória de 16 bytes para busca antecipada de instruções. Nessa memória, circuitos especiais lêem as instruções armazenadas e realizam uma antecipação das mesmas, de forma que, se elas forem efetivamente necessárias, já estarão previamente decodificadas [ROSCH, 90][HAYES, 89][INTEL, 90].

Em junho de 1988, a Intel produziu uma versão de baixo custo do 80386, o *Intel 80386SX*. Para evitar confusão, o modelo anterior foi denominado como *80386DX*. No 80386SX, o barramento de dados funciona internamente com 32 bits, porém externamente os dados são passados para um barramento de 16 bits. Dessa forma, dados de 32 bits são acessados como dois conjuntos de 16 bits. As linhas de endereço de memória real foram reduzidas de 34 para 25, permitindo que possam ser endereçados 16 megabytes de memória real. A finalidade da Intel foi concorrer com os fabricantes Haris e AMD, que estavam produzindo versões rápidas do 80286. Segundo testes realizados [NADEAU, 89], alguns PCs-286 realmente apresentaram um desempenho superior aos PCs-386SX. Entretanto, a grande vantagem dos microprocessadores 80386SX sobre

os 80286 é a possibilidade de executar todos os programas específicos do 80386, como o Unix V/386, Xenix-386, DESQview 386, Windows/386 entre outros [INTEL, 90].

Em 1989, o AT-486 foi implementado com o microprocessador 80486, também lançado em 1989 pela Intel. O 80486 caracterizou um grande avanço em relação ao 80386, apresentando as seguintes características adicionais:

- . Unidade Aritmética e Lógica inteira, compatível com a do 80386, porém altamente otimizada. Essa unidade possui uma linha de canalização (*pipelined*) com cinco estágios;

- . Registradores para busca antecipada de instruções (*prefetch buffers*);

- . Unidade completa de gerenciamento e proteção de memória;

- . Memória cache interna para dados e instruções de 8 kbytes. Tal memória apresenta um desempenho superior em relação às memórias cache externas, porque a transferência entre a cache e os registradores internos é otimizada por barramentos especiais;

- . Unidade de processamento em ponto flutuante compatível com o co-processador de ponto flutuante 80387. O fato do processador ser interno ao 486, permite que as transferências de

dados entre a unidade inteira e a de ponto flutuante possam ser otimizadas. A unidade de ponto flutuante pode trabalhar totalmente em paralelo com a de processamento de inteiros, o que não ocorre com o par 80386/80387.

A nível de instruções, o 80486 é totalmente compatível com os processadores da família 80x86 anteriores, apresentando, entretanto, 6 novas instruções, relacionadas com a manutenção do estado do cache, multiprocessamento e compartilhamento de estruturas de dados.

As primeiras versões do 80486 possuíam relógio interno com frequência de 25 Mhz, passando posteriormente para 33 Mhz. Visando aproveitar os projetos de placa mãe do AT-486 já existentes, a Intel lançou versões com relógio interno com frequência dobrada. Nesses sistemas, o processador realiza operações internas à plena frequência do relógio; contudo, em operações de acesso ao barramento externo (como acesso à memória e portas de entrada e saída) o processador trabalha com a metade da velocidade. Segundo essa filosofia, foram lançados os processadores 80486DX/2 de 50 e 66 Mhz, os quais, em operações externas, operam em 25 e 33 Mhz [INTEL, 90].

O elemento mais recente da família 80x86, por uma questão de estratégia de mercado, não saiu como 80586, e sim com o nome *Pentium* - o fato da Intel usar um "nome", permite o registro do mesmo, evitando seu uso pelos demais fabricantes. O Pentium foi lançado em março de 1993 e em maio já havia sistemas compatíveis com o IBM PC-AT utilizando o novo microprocessador.

As características adicionais do Pentium são apresentadas a seguir:

- . Unidade aritmética e lógica com cinco estágios de canalização (como a do 486), utilizando duas unidades em paralelo (o 80486 usa uma única unidade);

- . Os barramentos internos e externos são de 64 bits (no 486 são de 32 bits);

- . Caches de instrução e de dados separados, cada um com capacidade para 8 kbytes (o 80486 possui um cache único para dados e instruções de 8 kbytes);

- . Unidade para previsão de ramificações de instruções, onde também são armazenados os endereços das ramificações previstas (além dos registradores para busca antecipada de instruções);

- . Unidade de processamento em ponto flutuante organizada em linha de canalização com oito estágios, possuindo também circuitos dedicados para adição, divisão e multiplicação (muito mais complexa que a do 80486);

- . Organização superescalar, com possibilidade de executar duas instruções simultaneamente [RYAN, 93][HALFHILL, 93].

Comparando-se, em termos de desempenho, um 80486DX/2 de 66 Mhz com um Pentium também de 66 Mhz, e considerando que o compilador possa gerar código otimizado para o Pentium, de forma a tirar proveito de sua arquitetura, o Pentium obtém o dobro da velocidade do 80486DX/2 no processamento de inteiros e até cinco vezes em operações de ponto flutuante [ALPERT, 93].

Em março de 1994 a Intel lançou seu topo de linha: o *Pentium P54C*, um Pentium com relógio interno de 100 Mhz e com 200.000 transístores a mais que o Pentium convencional. Esses transístores extras são responsáveis pela divisão da frequência do relógio interno, gerenciamento de força e processamento duplo.

Além da Intel, outros fabricantes, como AMD e Cyrix, desenvolvem microprocessadores compatíveis com a linha 80x86. Até a geração do 80486, esses fabricantes utilizavam técnicas de engenharia reversa em seus projetos, ficando sempre uma geração atrás dos produtos da Intel, embora seus produtos, em alguns aspectos, apresentem desempenho superior aos fabricados pela Intel. Por exemplo, um 80286 de 20 Mhz da AMD é mais rápido que um 80386SX de 16 Mhz da Intel [HAYES, 89]. Na geração Pentium, tais fabricantes têm desenvolvido projetos próprios, com arquiteturas equivalentes à do Pentium, porém de concepção diferente, não mais utilizando técnicas de engenharia reversa. Esses novos microprocessadores, a exemplo do Pentium, executam todos os códigos de instruções da família Intel 80x86. Dentre os novos processadores compatíveis com a quinta geração da Intel tem-se o *Krypton-5* ou *K5* da AMD, o *M1* da Cyrix e o *Nx586* da NexGen [HALFHILL, 94a][HALFHILL, 94b]. Tal concorrência entre os fa-

bricantes resulta em benefícios de custo para os usuários e consumidores finais.

5.3. Critérios para Medida de Desempenho

A forma mais simplista para definir o desempenho de um computador é associar desempenho à velocidade de processamento. Entretanto, de uma maneira mais ampla, o conceito de desempenho está relacionado com a interação entre a linguagem de programação empregada e a arquitetura do computador, além do fator velocidade de processamento. Um dos responsáveis pela interação mencionada é o compilador, pois para uma dada linguagem de programação e uma dada arquitetura, apresentará um melhor desempenho o sistema que utilizar um compilador com a capacidade de tirar melhor proveito das características especiais da arquitetura da CPU.

A melhor maneira de se medir o desempenho de um computador é testá-lo diretamente na execução do programa aplicativo do usuário. Entretanto, nem sempre este procedimento pode ser executado. Quando um novo processador é lançado no mercado, é interessante que o usuário tenha meios de saber qual será o desempenho de seu programa aplicativo, sem ter realmente que testá-lo. Uma alternativa é recorrer a *testes padrões*, ou "benchmark" [WEICKER, 84], que fornecem índices para a avaliação de desempenho.

Um primeiro índice, um tanto quanto intuitivo, é a frequência do *relógio interno* do processador (*clock*). Quanto maior essa frequência, espera-se um maior desempenho. Contudo, em diferentes processadores, instruções com a mesma função podem requerer períodos diferentes do relógio, permitindo que processadores com menor frequência do relógio sejam mais eficientes.

Um índice mais apropriado, que leva em consideração o tempo de execução de uma instrução é o *MIPS* (Milhões de Instruções Por Segundo). O coeficiente MIPS deve ser considerado com um certo cuidado, visto que cada computador tem seu próprio conjunto de instruções, logo não tem muito sentido comparar a velocidade de execução das instruções. Um computador com filosofia CISC (*Complex Instruction Set Computer* ou Computador com Conjunto de Instruções Complexas) pode ter uma velocidade de execução de instruções mais baixa que outro baseado na filosofia RISC (*Reduced Instruction Set Computer* ou Computador com Conjunto de Instruções Reduzidas). Por exemplo, um processador 80486DX/2 de 50 Mhz (CISC) chega a 23 MIPS, enquanto que um processador Sun 4/200 Sparc de 33 MHZ (RISC) atinge 24 MIPS.

Um índice que é empregado, principalmente em computação científica, é a unidade para execução em ponto flutuante, medida em Milhões de Operações em Ponto Flutuante por segundo ou *Mflop/s* ou *Megaflop/s* ou ainda *Mflops*. Há também o índice *Linpack* [HOCKNEY, 88], medido em Mflops, que consiste em um programa em FORTRAN para a solução de 300 equações lineares usando técnicas matriciais.

Dois outros índices de desempenho, ou *benchmarks*, bastante empregados, são o *Whetstone* e o *Dhrystone*. O *Whetstone*, desenvolvido pela *Central Computer Agency* do Governo Britânico, é um programa sintético, que testa a computação numérica executando uma grande quantidade de operações aritméticas em ponto flutuante. Esse programa foi desenvolvido originalmente em ALGOL 60, sendo agora usado em FORTRAN [CURNOW, 76]. O *Dhrystone*, por sua vez, teve seu desenvolvimento voltado para a área de programação de sistemas, procurando levar em consideração a influência da linguagem, do compilador, do método de avaliação e as estruturas de dados envolvidas na linguagem [WEICKER, 84]. O *Dhrystone* tornou-se um padrão industrial, principalmente para a medida de desempenho de operações sobre números inteiros.

Um *benchmark* bastante aceito e suportado pela indústria é o *SPEC* (*System Performance Evaluation Consortium*) [GEE, 93]. O SPEC consiste em um conjunto de programas não triviais, escolhidos para padronizar a avaliação de desempenho, visando a obtenção de índices reais para a comparação de sistemas.

A versão *SPEC89* teve seu uso comprometido quando foi desenvolvido um compilador, o KAP da *Kuck and Associates*, que era otimizado para o componente *Matrix 300* da bateria de testes do SPEC89, de modo que o *Matrix 300* rodava em cache [WESTON, 94]. Acontece que o propósito do *Matrix 300* era testar o subsistema de memória sem cache. Com esse procedimento, o índice de desempenho deu um salto artificial. Este fato forçou o lançamento da versão *SPEC92*, que não inclui o *Matrix 300*.

O *SPEC92* consiste de 6 programas em linguagem C, que realizam um processamento massivo de inteiros, e 14 programas com intenso processamento em ponto flutuante, escritos em C e em FORTRAN. Os testes são executados na máquina alvo, somente com um processo de usuário ativo. Para cada programa o tempo de processamento é normalizado em relação ao tempo de processamento desse programa no DEC VAX11/780. Os índices *SPECint92* e *SPECfp92*, são obtidos através da média geométrica dos tempos de processamento normalizados de cada conjunto [GEE, 93].

5.4. Evolução do Desempenho da Família "IBM-PC"

A indústria de microeletrônica apresentou um significativo desenvolvimento nestas duas últimas décadas. Esse fato pode ser observado, comparando o lançamento do primeiro microprocessador, em novembro de 1971, o *Intel 4004*, de 4 bits, com capacidade para 60 mil instruções por segundo (0,06 MIPS) e integrando 1.900 transístores, com o lançamento, em março de 1993, do *Pentium*, com 64 bits, 150 MIPS e integrando 3.330.000 transístores. Como consequência desse desenvolvimento, surgiu a família de microprocessadores 80x86, que apresentou um salto em desempenho a cada nova geração.

A figura 5.1 ilustra a evolução da tecnologia na família 80x86, através da evolução na densidade de transístores.

De certa forma, a evolução da densidade de transístores implica em um aumento do desempenho, pois a diminuição das dimensões físicas dos transístores permite que se trabalhe com frequências de relógio (*clock*) maiores. O

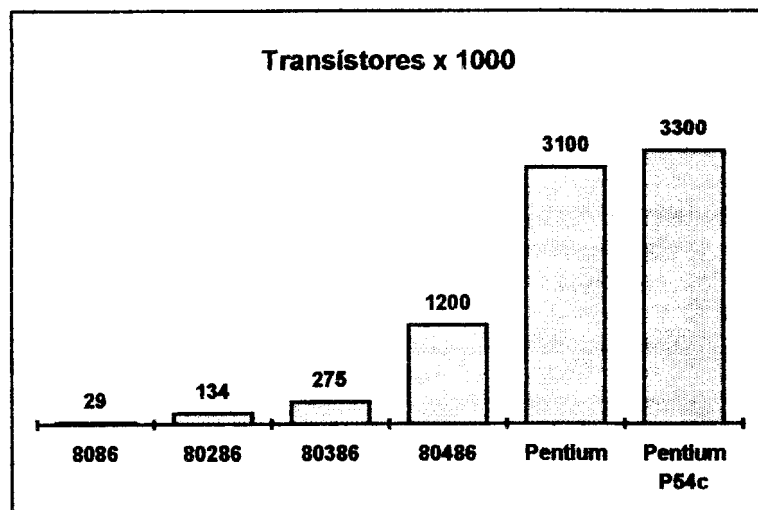


Figura 5.1 - Evolução da Densidade de transístores na família 80x86.

8086 trabalhava a uma frequência de 5 Mhz, com uma tecnologia de 3,0 microns (μm). Quando foi lançado, em 1986, o 80386 operava com uma frequência de 16 Mhz e empregava a tecnologia CMOS de 1,5 μm . Já em 1989, o 80486 saiu com uma frequência de 33 Mhz, com tecnologia CMOS de 1,0 μm [ALPERT, 93]. Finalmente, o Pentium foi desenvolvido com a tecnologia mais recente de semicondutores dos anos 90, a 0,8 μm , BiCMOS [RYAN, 90], operando inicialmente com 66 Mhz. Cabe observar que estes são valores de lançamento, pois a cada ciclo há um refinamento na tecnologia, que acaba levando a última geração de processadores a operar com frequência igual a inicial da próxima geração.

De forma a analisar a evolução da velocidade de processamento na família 80x86, inicialmente empregou-se o índice MIPS. Apesar das restrições já discutidas sobre o uso desse índice de desempenho, é coerente utilizá-lo aqui, porque todos os processadores foram projetados dentro da filosofia CISC. Além disso, cada nova geração 80x86 é completamente compatível, a

nível de conjunto de instruções, com todas as gerações anteriores.

A Figura 5.2 mostra a evolução na velocidade de execução das instruções. Quanto mais rápido o processador executar suas instruções, maior será a velocidade de execução dos programas aplicativos.

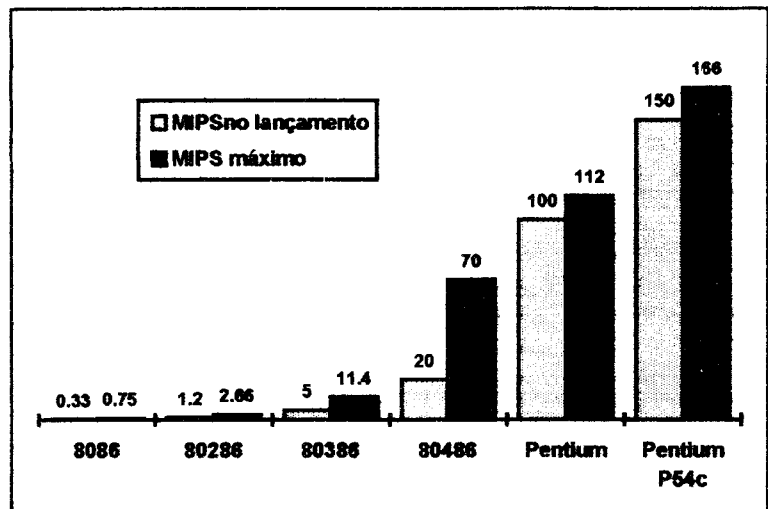


Figura 5.2 - Desempenho da linha 80x86 em MIPS.

Um resultado interessante é obtido dividindo-se o índice MIPS pelo número de transistores em milhões. O quociente obtido representa o número de instruções por segundo por transistor.

A figura 5.3 mostra os quocientes, para a linha 80x86. Observa-se que não é o Pentium o componente com maior quociente, mas sim o 80486 de última geração. Mesmo o 80386 de última geração supera

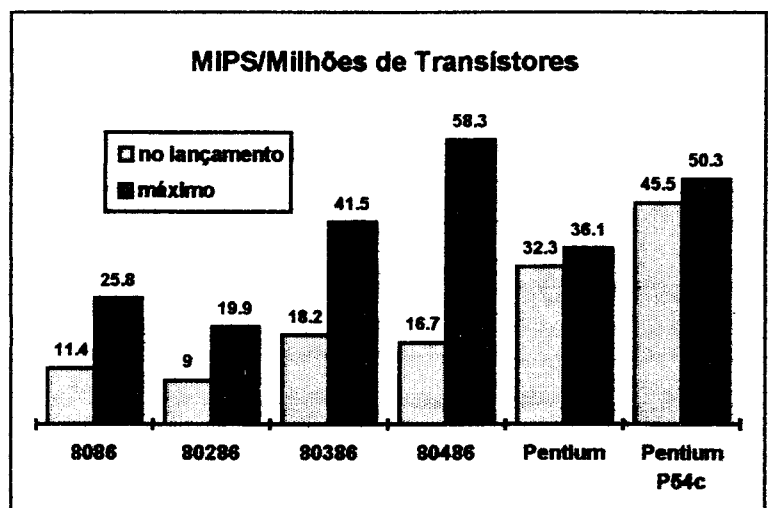


Figura 5.3 - Instruções por segundo por transistor para a família 80x86.

o Pentium tradicional, de ultima geração. Estes resultados indicam que a arquitetura mais otimizada é a do 80486. De onde se conclui: ou a arquitetura Pentium ainda não atingiu o seu auge, ou sua arquitetura superescalar não está sendo totalmente aproveitada.

5.4.1. Análise com Índices de Desempenho Padrões

Os índices de desempenho padrões (*benchmarks*), podem fornecer uma boa orientação para predizer o comportamento de um dado processador em um programa aplicativo, desde que se leve em conta as limitações associadas a esses índices.

Para essa análise de desempenho, foi escolhido um conjunto de microcomputadores da linha IBM-PC e seus compatíveis. O conjunto foi escolhido de maneira a abranger o maior número possível de elementos da família 80x86. Nessas máquinas foram executados programas que fornecem índices padrões. Foram usados o **Checkit**, versão 2.0, da *Touchstone Software Corporation*, e o **SI** (*System Information*), versões 4.5 e 5.0, de Peter Norton.

O **Checkit** é um dos programas mais completos para o teste de desempenho dos computadores da linha IBM-PC. Ele pode fornecer três coeficientes para a medida do desempenho. O primeiro é o coeficiente **Dhrystone**, o segundo é o coeficiente **Whetstone** e o terceiro é a velocidade de vídeo (dada em caracteres por segundo). Esse último índice não foi levado em conta nos teste efetuados, porque ele não é relevante para a aplicação do método

do gradiente conjugado ao controle de processos.

O *Norton SI* fornece três índices de desempenho: o *CI* (*Computing Index*), que mede o desempenho do processador em relação ao do processador do IBM-XT; o índice *DI* (*Disk Index*), que mede o desempenho do disco rígido em relação ao disco do IBM-XT; e o *OPI* (*Overall Performance Index*), que é o índice de desempenho total, sendo uma média ponderada dos índices anteriores. Destes três índices, considerou-se apenas o índice *CI* nos testes, já que, para os algoritmos de controle, interessa mais a velocidade do processador. Embora a Norton não divulgue o algoritmo ou o conjunto de programas que geram esses índices, optou-se por usá-los porque a maioria dos fabricantes de placas-mãe divulgam os índices Norton para propaganda de seus produtos. Os resultados dos testes são mostrados na tabela 5.1 (o símbolo (*) significa que o co-processador é interno ao microprocessador).

Observa-se, da tabela, que a evolução dos processadores traz consigo o aumento da velocidade de relógio (*clock*). O ganho no coeficiente *Drystone*, que está relacionado com as operações lógicas e inteiras, é inferior a 100, ao passo que o ganho no coeficiente *Whetstone*, que mede desempenho para processamento em ponto flutuante, é superior a 1000. Isto indica, que os processadores mais recentes têm o seu desempenho melhorado para operações em ponto flutuante.

O índice de desempenho *CI do Norton*, na versão 4.5, é praticamente insensível ao processador 80486. Ao se passar do

TABELA 5.1 - Valores de alguns Índices de Desempenho

Proces- sador	Co-pro- cessador	Clock Mhz	Drystone	(Mega) Whetsto- nes	Norton CI 4.5/5.0
8088	não	4,77	350	0,0067	1,0/1,0
8088	sim	4,77	344	0,1126	1,0/1,0
80286	sim	10	1888	0,2484	9,8/ ----
80386SX	sim	16	2985	0,7090	15,9/ ---
80386DX	não	40	11426	0,1262	47,7/10,9
80386DX	sim	40	11426	3,0785	47,7/10,9
80486DX	sim *	33	16164	5,7558	54,6/72,1
80486 DX/2	sim *	50	20897	7,7265	----/75,8

386DX (40 Mhz) para o 486DX (33 Mhz), o coeficiente *CI* varia de 47,7 para 54,6, enquanto que o coeficiente *Whetstone* praticamente dobra, na mesma situação. Na versão 5.0 do Norton, esse problema foi resolvido, porém a insensibilidade continuou ao se passar do 486DX para o DX/2. Como o Norton 4.5 é anterior ao 80486 e a versão 5.0 é posterior, deduz-se que o programa foi alterado para levar em consideração o desempenho do 486. Da mesma forma, como o Norton 5.0 é anterior à linha DX/2, é provável que versões mais recentes consigam medir o desempenho da linha DX/2. O fato da Versão 7.0 apresentar um "Compac 486DX/2 de 66 Mhz" como referência, com o índice 141,7, parece corroborar com este raciocínio. Fica difícil confiar em um índice de desempenho que deve ser

**TABELA 5.2 - Índices de Desempenho para Diversos
Microprocessadores Avançados**

CPU	Clock Mhz	Drysto- ne/s	(Mega) Whets- tons/s	Linpac MFLOP/s	MFLOPS	SPECint 92	SPECfp 92
486DX	33	16164 m	5,76 m	1,4 m	6,6 1	18,2 1	8,3 1
486DX/2	66	28*10 ³ e	10,2 e	2,5 e	11,7 e	32,3 1	16,0 1
Pentium	66	56*10 ³ e	36,3 e	8,9 e	41,6 e	64,5 1	56,9 1
Pentium P54C	90	76*10 ³ e	49,5 e	12,1 e	56,7 e	88,0 e	77,6 e
i860	40	85000 1	30,8 1	7,3 1	12,36 1	---	---
T800	30	---	4,6 1	---	2,25 1	---	---
SPARC 4/200	33	32000 1	7,2 1	1,1 1	2,6 1	---	---
PowerPC 604	80	---	---	---	---	75 1	85 1
MIPS R4200	40/80	---	---	---	---	55 1	30 1
T9000	50	---	---	---	25 1	---	---
ALPHA 21164	266/300	---	---	---	---	330 1	500 1

A tabela 5.2 foi montada com a finalidade de realizar uma comparação entre os processadores, sendo que alguns itens da literatura são dos próprios fabricantes, que tendem a superestimá-los. Nas medidas de desempenho realizadas por Tabak [TABAK, 90] para a linha 80386/387 foram obtidos índices de desempenho 20% inferiores aos fornecidos pela Intel. Os índices estimados foram obtidos de testes da literatura, que estavam normalizados em relação ao 80486DX/2 de 66 Mhz.

Os processadores i860, T800 e SPARC 4/200 são da época do 80386. O *i860* é um microprocessador vetorial, que na época de seu lançamento foi chamado de "*Cray on a Chip*" [FRIED, 91][MARGULIS, 89]. Possui 1 milhão de transístores, com tecnologia CHMOS IV e arquitetura RISC de *Harvard*, uma unidade de processamento de inteiros de 32 bits, e uma unidade de ponto flutuante com circuitos separados para adição e multiplicação, cada um com três estágios de canalização. A unidade de processamento de inteiros e o núcleo RISC podem trabalhar em paralelo com a de ponto flutuante. A unidade de ponto flutuante pode realizar uma adição e multiplicação no mesmo ciclo de máquina. O *i860* possui, também, uma memória *cache* separada para dados, com 8 kbytes e largura de 128 bytes, além de uma *cache* para instruções, com 4 kbytes e 64 bytes de largura. Além dessas características, o *i860* possui uma unidade para processamento gráfico tridimensional.

Do ponto de vista de desempenho, o *i860* chega aos 12,36 MFLOPS e, segundo o fabricante, em aplicações tipicamente vetoriais, pode chegar aos 80 MFLOPS de pico. Essas características, na época, fizeram com que alguns fabricantes de processadores paralelos migrassem do *T800*, que tem um desempenho inferior, para o *i860*.

O *T800* é o transputer da *Inmos*, que foi lançado em 1987. O *T800* é um microprocessador RISC de 32 bits, que opera em 30 Mhz. Possui uma RAM interna estática de 4 Kbytes, que facilita a implementação de uma pilha para variáveis locais, pois esta RAM pode ser acessada em um único ciclo de máquina. A unidade de

processamento em ponto flutuante também é incorporada à CPU e atinge 2,25 MFLOPS em 32 bits. O Transputer possui ainda 4 linhas bidirecionais (*Links*) seriais incorporadas a CPU, onde cada linha pode chegar até 30 Mbits/s em comunicação *full-duplex*. Isso resulta em uma capacidade total de comunicação, no chip, de 120 Mbits/s, ou seja, o equivalente a 12 linhas de redes Ethernet [STEIN, 88][POUNTAIN, 91].

A linguagem nativa do transputador é a *Occam* [JONES, 88], uma linguagem que permite a descrição e execução de processos em paralelo. Toda comunicação e sincronização dos processos é feita através de canais. A linguagem foi criada de forma a tirar proveito da arquitetura do transputer, principalmente da RAM interna, que é usada como pilha para a comutação de ambientes de programação entre os processos. Além disso, a alocação de processos pode ser feita no momento da compilação. Desse modo, quatro processos distintos, por exemplo, podem ser alocados a um único transputer, a fim de serem executados de maneira concorrente. Tais processos podem também ser alocados um a cada transputer, e também serem executados de maneira concorrente, com maior velocidade.

Apesar da vantagem das linhas bidirecionais e da facilidade de execução de vários processos em paralelo, características estas que facilitam muito a implementação de arquiteturas paralelas com múltiplos processadores, a grande desvantagem do T800 é seu baixo desempenho. Da tabela 5.2 pode-se observar que, em termos de desempenho, o T800 está um pouco abaixo do 80486DX de 33 Mhz. Além disso, seu custo e disponibilidade,

quando comparado com os 80486, não estimulam os projetistas a empregá-lo.

A arquitetura da *Sun SPARC*, por sua vez, não está ligada a nenhuma implementação específica de microprocessador - na realidade, há cinco fabricantes, que manufaturam produtos diferentes, e que implementam a arquitetura *SPARC*. O nome *SPARC* vem de *Scalable Processor Architecture*, ou seja, "Processador com Arquitetura Escalável". A filosofia da arquitetura escalável é fornecer um espectro amplo de implementações possíveis em termos de custo/desempenho, abrangendo desde microcomputadores a supercomputadores. A filosofia do projeto SPARC está baseada na filosofia *RISC* de *Berkeley* [TABAK, 90]. Como pode ser visto na tabela 5.2, o processador *Sun SPARC 4/200* possui desempenho comparável a um 80486DX/2 de 66 Mhz, ganhando deste último apenas no índice *Drystone*.

Da geração mais nova de processadores RISC, composta pelo *PowerPC 604*, o *MIPS R4000* e o novo transputer *T9000*, apenas o *PowerPC* apresenta desempenho ligeiramente inferior ao Pentium *P54C*. Ainda segundo a tabela 5.2, observa-se que o desempenho do *R4000* e do *T9000* [POUNTAIN, 91] situa-se entre 70 e 50%, respectivamente, do desempenho do Pentium *P54C*.

O último elemento da tabela 5.2, o *ALPHA 21164* da *Digital*, apresenta o maior desempenho de todos, chegando a 300 SPECint92 e 500 SPECfp92, com frequência de relógio interno de 300 Mhz. O *ALPHA* tem 9,3 milhões de transístores, com processo inicial de 0,5 μm e final de 0,35 μm ; seu encapsulamento é

cerâmico, com 499 pinos. A arquitetura é RISC, podendo realizar quatro instruções por ciclo do relógio, duas em ponto flutuante e duas inteiras. A *DEC* espera entrar com a produção em larga escala em março de 1995 [RYAN, 94].

A tabela 5.2, portanto, apresenta, claramente, a idéia do crescente desenvolvimento dos novos microprocessadores, onde, a cada geração, novas barreiras na escala de integração e velocidade são superadas.

5.5. Análise de Desempenho com o Método do Gradiente Conjugado

Conhecendo o desempenho de um programa aplicativo em um dado processador *A*, seu desempenho em outro processador *B* pode ser estimado, se índices de desempenho padrão, como aqueles discutidos na seção anterior, estiverem disponíveis para ambos os processadores. Para maior segurança, é interessante considerar um conjunto de índices, e não apenas um isolado, pois se a análise for baseada em um único índice, ela poderá ficar comprometida caso este índice possa ser corrompido, como ocorreu com o *SPEC89*.

Geralmente, índices que procuram medir velocidade de acesso à memória podem ficar comprometidos, se os programas que os calculam passarem a caber nas memórias *caches*, cada vez de maior capacidade nos novos processadores.

cerâmico, com 499 pinos. A arquitetura é RISC, podendo realizar quatro instruções por ciclo do relógio, duas em ponto flutuante e duas inteiras. A *DEC* espera entrar com a produção em larga escala em março de 1995 [RYAN, 94].

A tabela 5.2, portanto, apresenta, claramente, a idéia do crescente desenvolvimento dos novos microprocessadores, onde, a cada geração, novas barreiras na escala de integração e velocidade são superadas.

5.5. Análise de Desempenho com o Método do Gradiente Conjugado

Conhecendo o desempenho de um programa aplicativo em um dado processador *A*, seu desempenho em outro processador *B* pode ser estimado, se índices de desempenho padrão, como aqueles discutidos na seção anterior, estiverem disponíveis para ambos os processadores. Para maior segurança, é interessante considerar um conjunto de índices, e não apenas um isolado, pois se a análise for baseada em um único índice, ela poderá ficar comprometida caso este índice possa ser corrompido, como ocorreu com o *SPEC89*.

Geralmente, índices que procuram medir velocidade de acesso à memória podem ficar comprometidos, se os programas que os calculam passarem a caber nas memórias *caches*, cada vez de maior capacidade nos novos processadores.

**TABELA 5.3 - Controle Ótimo para Funções Contínuas com
20 Pontos no Intervalo de Discretização**

Processador	Co- processador	Clock (Mhz)	FC20 (Seg)
8088	não	4,77	680
8088	sim	4,77	80
80286	sim	10	28
80386SX	sim	16	16,25
80386DX	não	40	31,50
80386DX	sim	40	4,20
80486DX	sim *	33	3,10
80486DX/2	sim *	50	2,12

Também visando reduzir o tempo de computação, optou-se por realizar apenas 5 iterações do método do gradiente conjugado, pois como já discutido no *capítulo 3*, páginas III-29 e III-30, a diferença entre 5 e 10 iterações é pequena. A redução no tempo de computação ao se passar de 10 para 5 iterações no método é de aproximadamente 50%.

Dessa forma, os dados escolhidos para o teste foram:

- $W_i = 10000$; $R = 5$; $U_0 = 0$;
- 5 iterações do método;
- 20 intervalos de discretização no intervalo de operação $\Delta T_{op} = [0, 5]$ seg.

Os dados obtidos estão na tabela 5.3.

Pode-se observar que a partir do processador 80386DX de 40 Mhz, com coprocessador 80387, já é possível realizar o controle ótimo em tempo real utilizando o método de canalização, descrito no *capítulo 4*, porque os tempos de execução são menores que ΔT_{op} (5 segundos).

O FORTRAN 5.0 da *Microsoft* gera código para o co-processador, quando o mesmo está instalado na máquina. Além disso, ele permite uma otimização do código, para aproveitar as instruções especiais do 80286. Na realidade, esse compilador pode ser classificado como de 16 bits, visto que ele pode otimizar o código para máquinas de 16 bits.

Os dados da tabela 5.3 foram obtidos sem o recurso de otimização de código do compilador para efeito de comparação, pois o modo otimizado não pode ser executado no processador 8088. Já para os processadores 80386SX e 80486DX, o programa foi executado com otimização de código, cujos resultados são mostrados na tabela 5.4.

Para análise da convergência, o programa original mostrava na tela os valores do funcional de custo a cada iteração. Visando obter maior velocidade de execução, e considerando que em aplicações reais não há necessidade de visualização na tela, foi criada uma versão do programa onde a visualização dos dados na tela foi suprimida. Os resultados obtidos com essa versão também estão apresentados na tabela 5.4. Os testes foram

**TABELA 5.4 - Desempenho para o Caso de Funções Contínuas
para 20 e 200 Intervalos de Discretizações**

Processador	Normal (seg)		Otimizado (seg)		Otimizado s/video (seg)	
	FC20	FC200	FC20	FC200	FC20	FC200
80386SX + 80387 16 Mhz	16,25	101,40	11,53	53,50	6,32	44,16
80486DX 33 Mhz	3,10	19,29	1,15	5,50	0,49	4,66

realizados para 20 e 200 pontos de discretização no intervalo $\Delta T_{op} = 5$ seg, resultando nas colunas *FC20* e *FC200* na tabela 5.4.

Na tabela 5.4, pode-se notar que houve uma redução significativa no tempo de processamento para o 80486, devido à otimização do código. Já para o 80386SX, a redução maior foi devido à supressão da saída no vídeo.

5.5.1. Conclusão sobre o Desempenho do Método

Na tabela 5.4, pode-se observar que o menor tempo foi obtido para o 80486, com código otimizado, supressão da saída de vídeo e 20 pontos de discretização. Nesse caso, o tempo de execução foi de 0,49 segundos. Este valor não permite ainda o

controle com restrições severas de tempo, onde neste caso o tempo de execução deveria ser menor que o passo de integração, **dt**, no algoritmo de Runge-Kutta. Para este caso $dt=5/20$ ou 0.25 segundos. Portanto, o processador deve ter pelo menos o dobro da velocidade de processamento do 80486, de 33 Mhz, para que o controle do motor DC possa ser realizado em tempo real sem o uso do método de canalização.

Observando-se os índices da tabela 5.2, é possível verificar que o 80486DX/2 não fornece exatamente o dobro da velocidade do 80486DX de 33 Mhz. Desta forma, para que o método do gradiente conjugado pudesse ser aplicado, sem o uso do método de canalização, o processador deve ser pelo menos um Pentium convencional.

Não é necessário, entretanto, se empregar o método para o controle com restrições severas, pois, como já analisado no capítulo 4, o método de canalização fornece, qualitativamente, os mesmos resultados. Portanto, a partir de um 80386DX de 40 Mhz com co-processador já é possível realizar o controle ótimo em tempo real para o caso do motor DC.

Considerando a generalidade do método do gradiente conjugado, pode-se esperar que o controle ótimo em tempo real seja possível para qualquer processo que tenha constantes de tempo da ordem das do motor DC, com processadores que tenham pelo menos a potência computacional de um 80386DX de 40 Mhz, com co-processador.

CAPÍTULO 6

Arquiteturas Paralelas para Aplicações em Tempo Real

6.1. Introdução

Na solução de problemas de controle ótimo em tempo real pelo método do gradiente conjugado, o uso do processo de canalização (*pipeline*) leva a restrições brandas no tempo de computação. Nesse caso, a análise realizada no *capítulo 5* mostrou que um processador 80386DX de 40 Mhz e co-processador 80387 já é suficiente para a implementação do controle ótimo em tempo real, desde que os sistemas apresentem constante de tempo dominante maiores ou, no mínimo, iguais a do problema de teste.

Quando o sistema a ser controlado possui uma constante de tempo dominante menor que o caso de teste, ou for necessário utilizar um número maior de intervalos de discretização que os empregados nas análises do *capítulo 5* (Δt menor), os cálculos

devem ser realizados com uma velocidade maior. O aumento na velocidade pode ser obtido com o uso de um processador mais rápido ou dividindo-se a carga de trabalho entre vários processadores trabalhando em paralelo.

O desempenho dos diferentes processadores da família do IBM-PC, bem como o dos microprocessadores mais poderosos lançados recentemente no mercado, foi discutido no *capítulo 5*.

Neste capítulo será explorado o paralelismo implícito nos algoritmos do método do gradiente conjugado, visando a aplicação de uma arquitetura com múltiplos processadores no controle ótimo em tempo real. Serão analisadas três formas de implementação, a saber: com transputadores, solução via rede de computadores e uma proposta de arquitetura IBM-PC com compartilhamento de posições de memória.

6.2. Paralelização do Algoritmo do Gradiente Conjugado

O método do gradiente conjugado, utilizado para minimização de funções no \mathbb{R}^n , pode ser facilmente paralelizado, pois ele é constituído, basicamente, de produtos entre vetores e matrizes. O produto de uma matriz $(n \times m)$ por um vetor $(m \times 1)$, pode ser particionado como n produtos escalares de dois vetores $(m \times 1)$. Cada um destes n produtos escalares pode ser atribuído a um processador diferente, de forma a realizar o produto escalar

em paralelo. Um grau maior de paralelismo pode ser obtido particionando esses vetores do produto escalar em subvetores, e alocando cada subvetor a um processador diferente.

Esse tipo de paralelização tem sido usada por [CRONE, 93] e [EVANS, 93] na solução de sistemas de equações lineares, usando o *método do gradiente conjugado pré-condicionado*, uma variação mais recente do método do gradiente conjugado. Tais sistemas de equações lineares possuem um número muito grande de variáveis, geralmente resultantes de problemas de simulação tridimensional da dinâmica de fluídos. Segundo [CRONE, 93], o número de incógnitas pode ser da ordem de um bilhão, demandando um sistema de computação massivamente paralelo. Quando o número de incógnitas não é tão elevado, o número de processadores utilizados deve ser adequadamente escolhido, para evitar que a comunicação entre os processadores acabe sobrecarregando o processamento.

Quando o método do gradiente conjugado é aplicado na solução dos problemas de controle ótimo, tem-se o caso de otimização com dimensão infinita (o espaço envolvido é o de *Hilbert*). Embora a dimensão do espaço seja infinita, o número de variáveis envolvidas é pequeno, geralmente uma ordem a mais que a dimensão das equações diferenciais que descrevem o sistema. Por exemplo, o motor DC foi modelado por uma equação diferencial de terceira ordem, logo o sistema resultante para a solução pelo método do gradiente conjugado tem quatro variáveis de estado. A quarta variável é a que foi introduzida para facilitar o cálculo do funcional de custo. Este caso é de otimização com dimensão infi-

nita, porque cada elemento da matriz de estado é uma função contínua do tempo, que, no processamento numérico, deve ser discretizada e subdividida em um número adequado de intervalos.

Na paralelização deste problema deve-se considerar o fato de que cada variável de estado é uma função contínua do tempo, que, ao ser discretizada, seu valor no instante t_{k+1} depende do valor anterior em t_k . Desta forma, não é possível utilizar mais de um processador para uma mesma variável de estado, devido a sua dependência com o instante anterior. Então uma solução é alocar um processador para cada variável de estado do sistema. Esta abordagem leva a um número pequeno de processadores trabalhando em paralelo, número este que no máximo é igual ao número de variáveis de estado do sistema a ser controlado mais um. Portanto, no caso do motor DC, quatro processadores são suficientes.

6.2.1. Abordagem Computacional

Considerando a alocação de um processador para cada variável de estado, será feita uma revisão no algoritmo de controle ótimo pelo método do gradiente conjugado, apresentado no *capítulo 3*, para identificação dos trechos no algoritmo passíveis de paralelização.

O algoritmo do gradiente conjugado foi desenvolvido nas etapas **(1)** à **(2.3)** da página III-20, *capítulo 3*, transcritas a seguir:

- (1) Partindo de uma estimativa \mathbf{u}_0 , calcula-se o gradiente $\mathbf{g}(J(\mathbf{u}_0)) = \mathbf{g}_0$ e toma-se um vetor \mathbf{p}_0 , tal que $\mathbf{p}_0 = -\mathbf{g}_0$ (equação (2.6.3-4));
- (2) Para $i = 0, 1, \dots, n$, executa-se os seguintes passos:
 - (2.1) Determina-se, por meio de uma busca unidimensional, o escalar $\alpha_i > 0$, que minimiza $J(\mathbf{u}_i + \alpha_i \mathbf{p}_i)$ (equação (2.6.3-6));
 - (2.2) Calcula-se $\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha_i \mathbf{p}_i$ (equação (2.6.3-5));
 - (2.3) Obtém-se o novo \mathbf{p}_{i+1} tal que $\mathbf{p}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{p}_i$ (equação (2.6.3-7)), com β_i dado por (2.6.3-8) e $\mathbf{g}_{i+1} = \mathbf{g}[J(\mathbf{u}_{i+1})]$.

As operações de produto escalar, adição e subtração de vetores, descritas explicitamente neste algoritmo não serão paralelizadas, sendo executadas sequencialmente por um único processador, pois representam uma parcela muito pequena do esforço computacional de todo o algoritmo. No algoritmo acima, a quase totalidade do esforço computacional é empregada no cálculo do gradiente e na busca unidimensional.

O cálculo do gradiente para uma dada entrada de controle $u(t)$, como pode ser observado no algoritmo da página III-19, capítulo 3, requer que as equações de estado sejam integradas

de t_0 a t_f , para que a condição terminal sobre a equação adjunta seja conhecida. Partindo da condição terminal, a equação adjunta deve ser integrada de t_f a t_0 , pois o gradiente é expresso em função da solução dessa equação adjunta.

O algoritmo da busca unidimensional, vide *seção 3.4.2, capítulo 3*, envolve o cálculo do funcional de custo a cada passo. Por sua vez, o funcional de custo é calculado em função das variáveis de estado no tempo final. Desta forma, cada passo de procura do ótimo no algoritmo da busca unidimensional envolve a integração das equações de estado do sistema de t_0 a t_f .

Considerando o exposto nos parágrafos anteriores, pode-se deduzir que a integração das equações de estado e das equações adjuntas representam a maior parte do esforço computacional envolvido no algoritmo de controle ótimo pelo método do gradiente conjugado. Portanto, para se obter um melhor desempenho, é nas rotinas de integração que a paralelização deve ser aplicada.

6.2.2. Paralelização da Rotina de Integração

Na integração das equações de estado e das equações adjuntas é empregado o algoritmo de *Runge-Kutta* de quarta ordem, usando o *método de Gill* [RALSTON, 76], como discutido no *capítulo III, seção 3.4.1*.

No caso do motor D.C., a implementação em Fortran da rotina de integração é realizada pelo subprograma "RUNGE", lista-

do no *Apêndice B.1*. O algoritmo de *Runge-Kutta* requer que a função a ser integrada seja dada em uma forma de equação a diferenças, que é realizada pelos subprogramas "FUNX" e "FUNL", listados no *Apêndice B.2*. O subprograma "FUNX" define a equação a diferenças para as equações de estado do sistema, ao passo que "FUNL" faz o mesmo para as equações adjuntas.

Visando identificar os trechos paralelizáveis do algoritmo de integração, a sub-rotina "RUNGE" é transcrita a seguir:

C SUBPROGRAMA RUNGE

```

SUBROUTINE RUNGE(T,H,XOUL,UDT)
DIMENSION F(11),UDT(201),Y(12),AK(12)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
EQUIVALENCE(Y(2),X1(1)),(AK(2),F(1))
C WRITE(*,900)
900 FORMAT('RUNGE')
Y(1)=T
AK(1)=1.
DO 10 J=1,4
IF(XOUL)7,7,8
C XOUL=0 PARA FUNX
C XOUL=1 PARA FUNL
7 CALL FUNX(T,H,F,UDT)
GO TO 9
8 CALL FUNL(T,H,F,UDT)
9 DO 10 I=1,N11
Z=A(J)*(AK(I)-B(J)*Q(I))
Y(I)=Y(I)+H*Z
10 Q(I)=Q(I)+3*Z-C(J)*AK(I)
RETURN
END
```

Nessa listagem, o trecho em negrito é realizado N_{11} vezes para cada valor da variável J , onde N_{11} é o número de equações a serem integradas; no caso do problema de teste, $N_{11} = 4$. Isso significa que, cada vez que se entra no laço "DO" definido na linha 9, o trecho em negrito é executado N_{11} vezes. Além disso, as execuções do trecho em negrito, dentro do laço "DO" são independentes e podem ser realizadas em paralelo.

Assim, a abordagem de paralelização proposta é o uso de N_{11} processadores para a execução do trecho em negrito dentro do laço "DO" da linha 9. Esta abordagem implica em que cada processador deve passar o valor calculado de $Y(I)$ e de $Q(I)$ para os demais.

É interessante lembrar que, na solução dos problemas de controle ótimo pelo método do gradiente conjugado, se o sistema é de ordem n , então o sistema de equações diferenciais a ser resolvido tem $(n + 1)$ equações, devido à adição do termo que fornece a integral do funcional de custo (vide **seção 3.2**: modelamento do motor). Portanto, podem ser empregados, segundo a abordagem proposta, $(n + 1)$ processadores em paralelo.

Ainda no subprograma "RUNGE", pode-se observar que são chamados os subprogramas "FUNX" e "FUNL", que definem as equações a diferenças para as equações de estado e adjuntas, respectivamente. As listagens desses subprogramas, para o caso do motor D.C., são apresentadas a seguir:

C **SUBPROGRAMA FUNX**

```
SUBROUTINE FUNX(T,H,F,UDT)
DIMENSION F(11),UDT(201)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
```

C

C COMMONS DO USUARIO

C

```
COMMON R, W(3)
I=(T-TI)/DT+1
F(1)=X1(2)
F(2)=-X1(2)/3.+5.*X1(3)
F(3)=-X1(3)+.1*UDT(I)
F(4)=(X1(1)-10)**2+R*UDT(I)**2
RETURN
END
```

C **SUBPROGRAMA FUNL**

```
SUBROUTINE FUNL(T,H,F,UDT)
DIMENSION F(11),UDT(201)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
```

C

C COMMONS DO USUARIO

C

```
COMMON R,W(3)
```

C

```
I=(T-TI)/DT+1
F(1)=-2*(X(1,I)-10)*X1(4)
F(2)=-1*(X1(1)-1./3.*X1(2))
F(3)=-1*(5*X1(2)-X1(3))
F(4)=0
RETURN
END
```

No subprograma "FUNX", as 4 linhas em negrito podem ser atribuídas a processadores diferentes, e executadas em paralelo, o mesmo ocorrendo com as 4 linhas em negrito do subprograma "FUNL". Entretanto, os valores calculados em cada processador devem ser passados para os demais. O número de equações a diferenças calculadas em cada um destes subprogramas e, conseqüentemente, atribuídas a cada processador, é $N_{l1} = n+1$, que, no caso do motor D.C., é 4.

6.3. Implementações da Arquitetura Paralela

Na seção anterior foi determinado o número de processadores a serem empregados em paralelo na solução o problema de controle ótimo pelo método do gradiente conjugado. Também foi explicitado que os processadores devem compartilhar o resultado de seus cálculos com os demais.

Considerando que a arquitetura proposta será aplicada no caso do controle ótimo em tempo real do motor D.C., serão consideradas arquiteturas com 4 processadores.

A forma como os processadores compartilham os seus dados define o tipo de arquitetura a ser implementada. Uma possível solução é o uso de uma rede de transputadores, onde suas linhas bidirecionais (*links*) seriam utilizadas para realizar o compartilhamento dos cálculos. Outra forma de arquitetura paralela é o emprego de microcomputadores tipo IBM-PC, onde o comparti-

lhamento dos dados é feito via uma rede tipo Ethernet. Finalmente, uma terceira arquitetura, que também emprega microprocessadores tipo IBM-PC, com compartilhamento dos dados via registradores de memória comuns aos processadores.

6.3.1. Tempo de Processamento X Tempo de Transmissão de Dados

Na implementação da arquitetura paralela, os processadores devem realizar os cálculos especificados nas linhas em negrito nas listagens dos subprogramas "RUNGE", "FUNL" e "FUNX". Analisando tais linhas, pode-se observar que no máximo o processador deve realizar 6 operações em ponto flutuante para cada resultado que ele deve passar. Esse resultado é um número em ponto flutuante de 32 bits. Para analisar a eficiência das arquiteturas propostas é necessário verificar a relação entre o tempo que os processadores levam para calcular um resultado e o tempo necessário para passar este dado aos outros processadores. A implementação da arquitetura paralela é eficiente somente se o tempo de transmissão dos resultados for uma pequena fração do tempo de processamento.

6.3.2. Rede de Transputadores

Uma possível implementação da arquitetura paralela é com o uso de transputadores como elementos básicos de processamento. Neste caso, pode-se tirar proveito dos 4 *links*, ou linhas bidirecionais, que cada transputador contém. Um esquema dessa

arquitetura pode ser visto na figura 6.1, onde cada transputador está conectado aos outros três restantes através dos *Links*.

Para calcular a eficiência dos canais de comunicação, é necessário determinar a velocidade de transmissão de dados nos *links*. Cada *link* bidirecional é composto de uma linha serial para a transmissão e outra para a recepção, podendo trabalhar no modo *full duplex*. Segundo os dados da literatura [INMOS, 89], o transputador utiliza 11 bits para transmitir um *byte* de dados. O formato é um *start bit* seguido de um *bit* com o valor 1 mais todo o byte de dados e um *stop bit*. O receptor, por sua vez ao receber

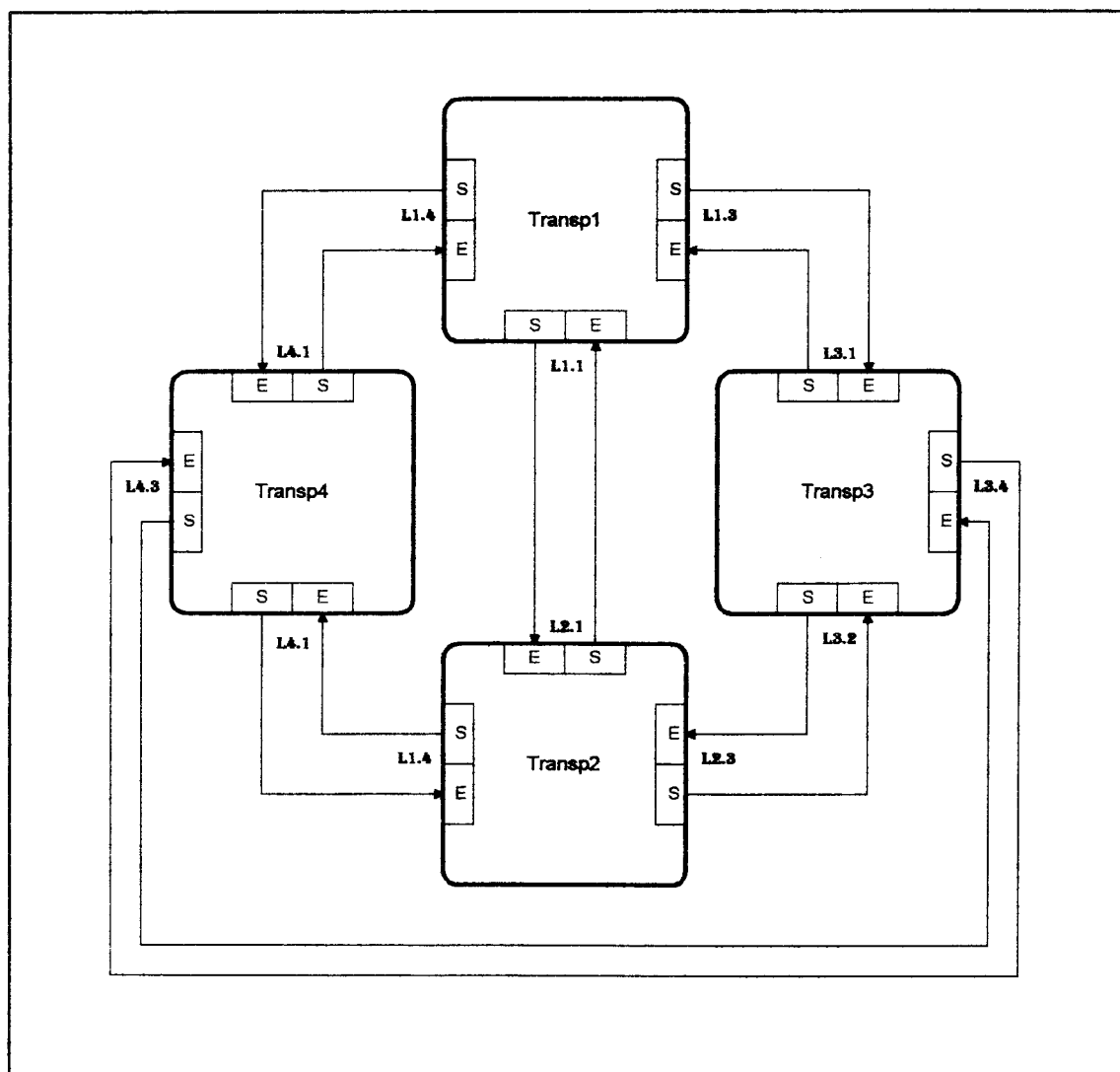


Figura 6.1 - Implementação da arquitetura paralela com transputadores.

um dado, envia dois bits de reconhecimento para indicar que o dado foi recebido e que o registrador de entrada *buffer* está pronto para receber outro *byte*. Os *bits* de reconhecimento se constituem da cadeia "10".

O transputador T800 pode operar com taxas de 10 e 20 Mbits/seg em seus *links*. Devido aos *bits* extras de controle e a espera de reconhecimento, como era de se esperar, as taxas de transmissão efetiva de dados são menores. As seguintes taxas são fornecidas para o T800 [INMOS, 89]: a 10 Mbits/seg, resulta em 910 Kbytes/seg no modo unidirecional e 1.250 Kbytes/seg no modo bidirecional; a 20 Mbits/seg, pode-se chegar a 1.740 Kbytes/seg no modo unidirecional e a 2.350 kbytes/seg no modo bidirecional. Ainda, segundo a literatura, estas taxas sofrem pouca degradação quando os 4 *links* são operados simultaneamente.

Os dados fornecidos pela *Inmos* para a operação em 10 Mbits/seg, no modo unidirecional, foram confrontados com dados obtidos experimentalmente, e não houve discrepância dos resultados. Assim sendo, pode-se supor que os demais valores fornecidos pelo fabricante [INMOS, 89] também estão corretos.

A análise da eficiência dos canais de comunicação requer o cálculo do tempo que os processadores levam para transmitir o dado calculado e receber os dados de todos os outros processadores. Como os 4 *links* operam simultaneamente em cada transputador, no modo bi-direcional, ao mesmo tempo em que um transputador está enviando seu dado ele está recebendo o dado dos outros três. Logo, considerando um número em ponto flutuante de

32 bits, o tempo total t_t para que todas as transmissões ocorram é de $(32/8 * 1/1.250.000)$ segundos, ou seja, 3,2 μ secs, com os *links* operando a 10 Mbits/seg. Considerando a operação dos *links* a 20 Mbits/seg, o tempo para as transmissões é de $(32/8 * 1/2.350.000)$ segundos ou $t_t = 1,7$ μ secs.

Definindo a razão entre *tempo de transmissão/tempo de processamento*, denotada por R_{tp} , como:

$$t_t/t_p = R_{tp} \quad (6.1)$$

onde t_t é o tempo de transmissão do dado e t_p é o tempo que um processador leva para realizar as operações em ponto flutuante.

Considerando a tabela 5.2, pode-se verificar que o T800 apresenta um desempenho de 2,25 MFLOPS e, conseqüentemente, o tempo para realizar as 7 operações em ponto flutuante, discutidas na seção 6.3.1, é de $(7/2.250.000)$ ou $t_p = 3,11$ μ secs. Assim, considerando os valores de t_t já calculados, temos:

- $R_{tp} = 3,2/3,11 = 1,03$, para os links operando a 10 Mbits/seg e
- $R_{tp} = 1,7/3,11 = 0,55$, quando os links operam em 20 Mbits/seg.

A eficiência da arquitetura paralela proposta pode ser melhor analisada definindo o coeficiente de eficiência η , e o ganho de velocidade da arquitetura em relação a um único processador: G_p .

O coeficiente η é dado por:

$$\eta = t_p / (t_p + t_t) \quad (6.2)$$

e o ganho de velocidade G_p é:

$$G_p = \eta * n \quad (6.3)$$

onde n é o número de processadores em paralelo.

A velocidade V_p da arquitetura, em Mflops, é dada por:

$$V_p = G_p * V \quad (6.4)$$

onde V é a velocidade, em Mflops, de um único processador da arquitetura paralela.

A equação (6.2) pode ser reescrita como:

$$\eta = t_p / (t_p + t_t) = 1 / (1 + t_t / t_p) = 1 / (1 + R_{tp}) \quad (6.5)$$

Das equações (6.5) e (6.3) os valores de η e G_p podem ser calculados para as velocidades dos links de 10 e 20 Mbits/seg. Para 10 Mbits/seg $\eta = 0,493$ e $G_p = 1,97$. Este valor de G_p significa que o ganho de velocidade obtido com 4 transputadores é apenas 97% superior ao ganho obtido com apenas um deles. Com os links operando a 20 Mbits/seg resulta em $\eta = 0,645$ e $G_p =$

2,58.

Neste caso, o ganho de 2,58 significa que, com os 4 transputadores, o ganho em velocidade de processamento é de 158% em relação a um único transputador.

Conclusão sobre a Implementação com Transputadores

Considerando a equação 6.2, pode-se observar que o coeficiente de eficiência aumenta a medida que diminui o tempo gasto na transmissão dos dados entre os processadores. No caso da solução com transputadores, uma vez que os *links* operam na capacidade máxima (20 Mbits/seg), a velocidade de processamento com 4 transputadores quase chega a ser 2,6 vezes a velocidade obtida com um único transputador.

O desempenho dessa arquitetura não pode ser considerado bom, pois, para esse caso, o coeficiente η está próximo de 0,6, quando uma arquitetura ideal leva a um coeficiente próximo de 1 e, conseqüentemente, a um ganho de velocidade de quatro vezes.

Outro fator que não estimula o uso de transputadores é seu baixo desempenho perto de outros processadores atualmente disponíveis. Na tabela 5.2, pode-se observar que o T800 atinge 2,25 MFLOPS, resultando em uma velocidade de 5,81 MFLOPS para os quatro operando em paralelo, com *links* a 20Mbits/seg. Este valor é muito pequeno quando se considera a velocidade de 6,6 MFLOPS de um 80486DX/33 (tabela 5.2). Em outras palavras, um único 80486DX/33 é mais rápido que a arquitetura paralela implementada

com 4 transputadores T800.

Mais recentemente a *Inmos* anunciou o lançamento de um novo modelo de transputador, o *T9000*, capaz de atingir 25 MFLOPS de velocidade de processamento e taxa de transferência nos *links* de 100 Mbits/seg [FLETCHER, 93]. Entretanto, o aumento de aproximadamente 10 vezes, tanto na velocidade de processamento como na taxa de transmissão, significa que o T9000 vai apresentar o mesmo R_{tp} do T800 e, conseqüentemente, o mesmo coeficiente de eficiência, de aproximadamente 0,6. Este valor do coeficiente de eficiência significa que, se a arquitetura paralela for implementada com os processadores T9000, a velocidade resultante será de aproximadamente 58 MFLOPS. Segundo a tabela 5.2, um único processador *Pentium* com 66 Mhz de frequência de *clock* atinge 41,6 MFLOPS, e o Pentium P54C operando a 90 Mhz chega aos 56,7 MFLOPS.

Da análise realizada nesta seção, pode-se concluir que a solução com transputadores não é recomendada, pois, mesmo a implementação com 4 processadores T9000, apresenta desempenho semelhante ao atingido com um único processador *Pentium P54C*.

A solução com transputadores apresenta ainda uma outra desvantagem: o fato de não ser uma solução aberta. Os transputadores são produzidos apenas pela *Inmos*, uma divisão da *SGS (Thomson Microelectronics)*. A distribuição é deficiente e eles não são encontrados com facilidade, além do que ocorreram sucessivos atrasos no lançamento do T9000. Ainda, não existe um microcomputador de arquitetura "aberta de fato" implementado com transputadores, como ocorre com a linha de microprocessadores da Intel,

que é suportada pelos computadores *IBM-PC*. Como consequência, o número de programas aplicativos disponíveis para sistemas baseados em transputador é pequeno, comparado com os disponíveis para a linha *Intel/IBM-PC*.

6.3.3. Solução via Rede de Computadores

Uma forma de aproveitar a arquitetura aberta dos computadores *IBM-PC* e desfrutar de todos os programas aplicativos disponíveis para essa linha, bem como dos microprocessadores avançados lançados pela Intel (80486, Pentium e outros que já estão em projeto, como o P6), seria utilizar estes computadores como elementos básicos da arquitetura paralela.

Uma possível forma de implementação da arquitetura paralela com a linha *IBM-PC* é através do compartilhamento dos dados entre os processadores via uma rede tipo *Ethernet*. Esta implementação tem a vantagem da tecnologia de redes estar bastante desenvolvida, existindo muitos fabricantes de circuitos e programas aplicativos para a linha *IBM-PC*.

A arquitetura proposta para esta implementação está esquematizada na figura 6.2, onde os 4 processadores são interligados por um barramento no padrão *Ethernet*.

Avaliação do desempenho

Da mesma forma que na arquitetura com transputadores,

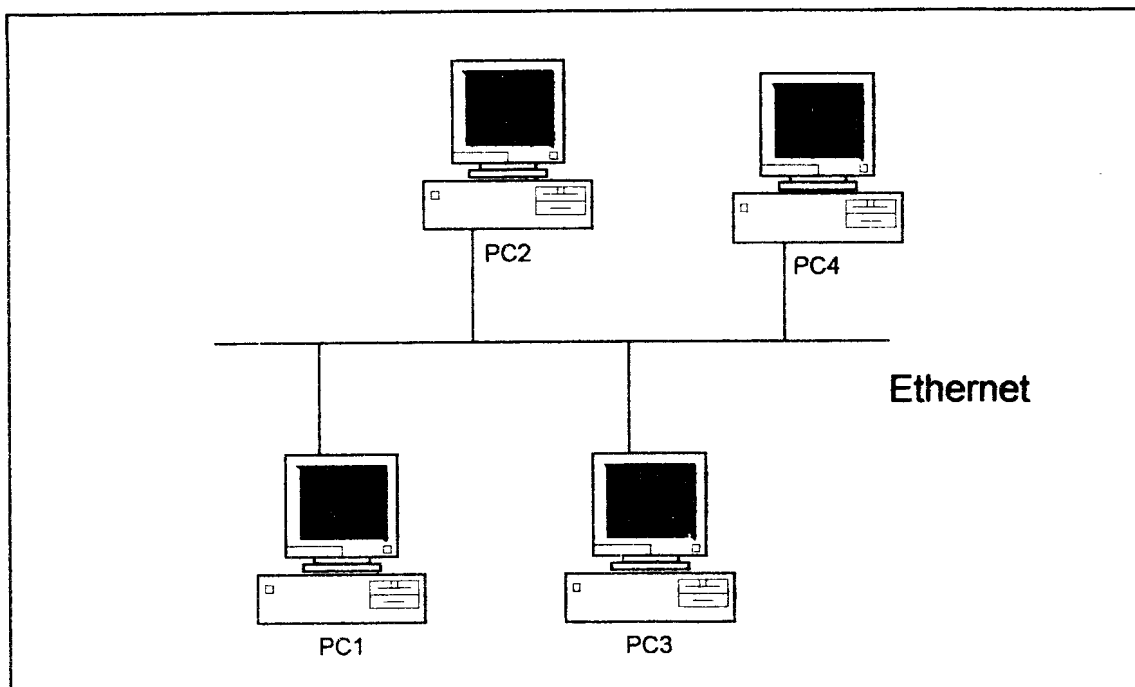


Figura 6.2 - Implementação da arquitetura paralela usando rede *Ethernet*.

o desempenho vai depender das taxas de comunicação que pode ser atingida com a rede. Segundo a literatura [TANENBAUM, 88], a rede *Ethernet* pode chegar aos 10 Mbits/seg. Entretanto, é necessário determinar a taxa efetiva de transferência dos dados, onde os *bits* de controle e cabeçalhos não devem ser levados em conta.

Uma primeira tentativa foi a medida da taxa de transferência entre dois processadores 80486, conectados via rede *Ethernet*, ambos com o sistema operacional Unix, utilizando o serviço FTP do conjunto de protocolos TCP/IP.

Os protocolos TCP/IP foram desenvolvidos pela *Agência de Projetos de Pesquisa Avançados de Defesa (DARPA - Defense Advanced Research Projects Agency)*, como parte de um projeto de interconexão de redes heterogêneas. Esta tecnologia acabou rece-

bendo o nome de *Internet*. A *Internet* provê um conjunto de protocolos que permite que redes físicas diferentes sejam interconectadas, de maneira transparente quanto ao tipo de aplicação e tipo dos dados. Na *Internet*, os dois principais protocolos, o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*) acabaram dando nome a arquitetura, que ficou conhecida como arquitetura TCP/IP de interconexão de redes [COMER, 91].

No sistema operacional Unix, os protocolos TCP/IP foram escolhidos para implementação do subsistema de comunicação de dados e, portanto, os protocolos TCP/IP estão disponíveis em todo sistema Unix adequadamente instalado.

A transmissão de dados pelo serviço FTP apresentou uma taxa variável, segundo o tipo e tamanho dos arquivos a serem transferidos. Para a transmissão de 1 *byte* de texto, a taxa de transmissão foi de 9,8 Kbytes/seg. Esta taxa subiu para 10,2 Kbytes/seg na transmissão de 4 bytes de texto. O tamanho de 4 bytes é importante porque os números em ponto flutuante, com precisão simples, que devem ser passados entre os processadores, ocupam quatro *bytes*.

O aumento da taxa de transmissão, de maneira proporcional à quantidade de *bytes* transferidos, indicava que os protocolos de comunicação introduziam *bytes* extras, na forma de cabeçalhos, endereços de rede e caracteres de controle.

Para verificar tal dependência, foi realizado um teste com um arquivo de aproximadamente 640 Kbytes de tamanho, conside-

rando, inicialmente, o arquivo como de caracteres em ASCII, e, posteriormente, considerando o arquivo como binário. Como resultado, obteve-se as seguintes taxas de transferência para o FTP: 108 Kbytes/seg para o arquivo ASCII e 174 Kbytes/seg para o arquivo binário. Estas diferenças confirmam a dependência da velocidade de transmissão com os *bytes* extras para endereçamento, controle e correção de erros.

O serviço FTP funciona sobre o protocolo TCP, o qual é considerado um protocolo confiável, sendo dotado de mecanismos de detecção e correção de erros, os quais introduzem muitos *bytes* extras no conjunto de dados a ser transferido, inclusive implicando na retransmissão de grupos de dados não recebidos ou não confirmados. Por sua vez, o protocolo TCP funciona sobre o protocolo IP, que é baseado em datagramas. O cabeçalho de um datagrama

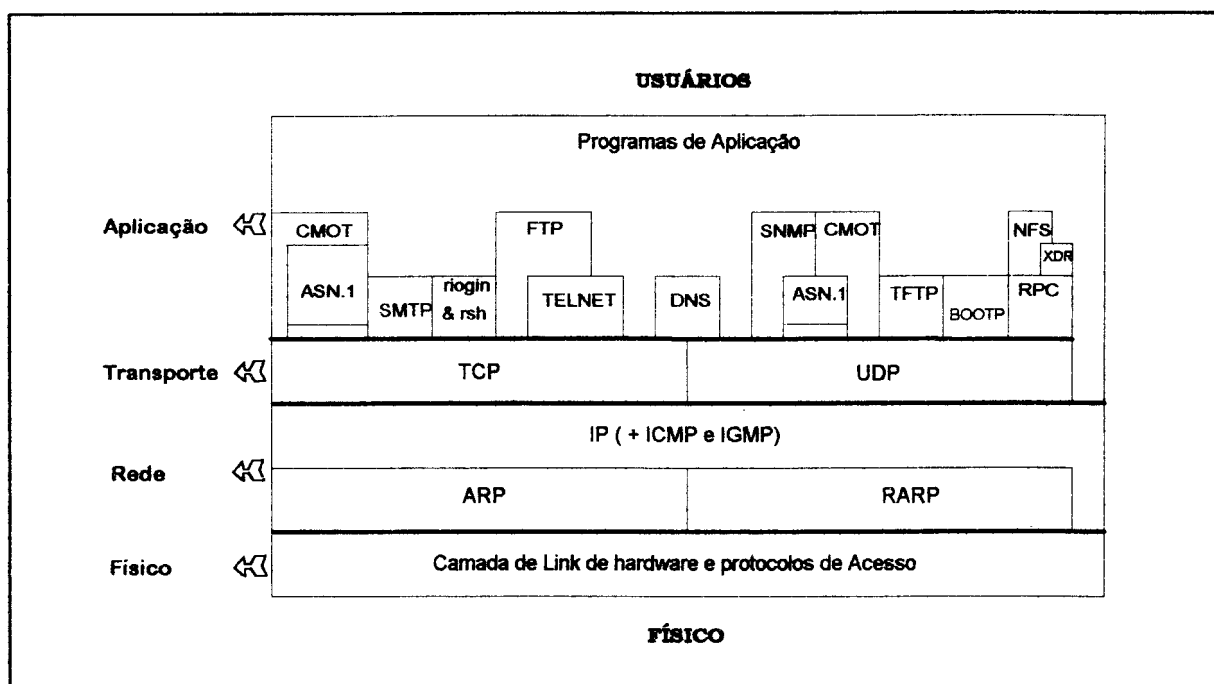


Figura 6.3 - Arquitetura da Internet.

IP típico contém cerca de 24 *bytes*. Finalmente, o datagrama IP deve ser encapsulado em um ou vários *frames* (quadros) no nível mais inferior (*Ethernet*). A figura 6.3 apresenta um esquema da arquitetura *Internet*, com o encadeamento em camadas dos protocolos e serviços disponíveis na *Internet*. O próprio *frame Ethernet* também possui seu cabeçalho e *bits* de CRC (*Cyclic Redundance Check*), como mostrado na figura 6.4.

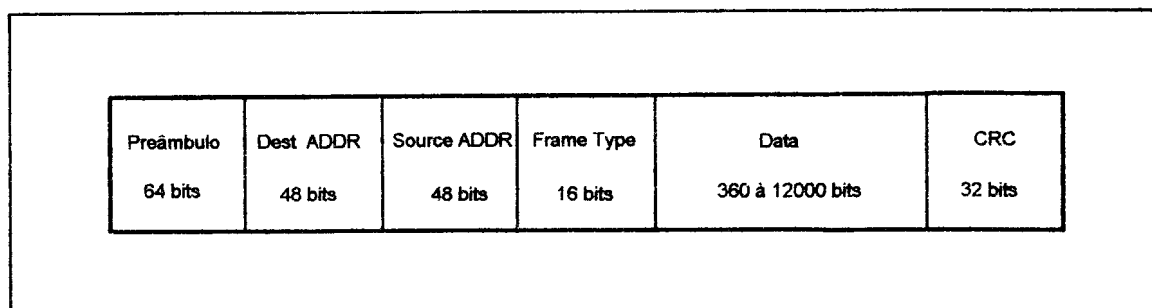


Figura 6.4 - Frame Ethernet.

Como consequência do encadeamento de todos esses protocolos e camadas, a transmissão dos dados fica muito lenta, inviabilizando o uso da rede como forma de transmissão de dados na arquitetura paralela.

Uma forma de ganhar mais velocidade seria transmitir os dados pelo protocolo *Ethernet*, uma vez que este constitui a camada mais próxima do nível físico. Para isso, é necessário um programa para enviar os dados que se quer transmitir diretamente à placa *Ethernet*. Esse programa, no entanto, depende da placa de rede empregada; as variações de fabricantes e da placa de rede são relevantes, uma vez que o endereço físico *Ethernet* difere de

uma placa para outra. Outra característica do programa mencionado é que ele seja suficientemente pequeno, pois caso contrário a velocidade de 10 Mbits/seg nunca será atingida.

O coeficiente de eficiência de transmissão dos dados na rede *Ethernet* η_{ts} é definido como:

$$\eta_{ts} = Q_{bd}/Q_{bt} \quad (6.6)$$

onde Q_{bt} é a quantidade total de *bits* transmitida no *frame* e Q_{bd} é a quantidade de *bits* de dados transmitida no *frame*.

Pode-se observar que o coeficiente de eficiência de transmissão η_{ts} depende da quantidade de dados a ser transmitida: quanto maior a quantidade de dados, maior o valor deste coeficiente.

É interessante notar que o valor máximo de η_{ts} é obtido quando a quantidade de *bits* de dados transmitida é igual ao máximo que um *frame Ethernet* pode transportar, que, segundo a figura 6.4, é

$$\eta_{ts(max)} = 12,000 / (64+48+48+16+12.000+32) \approx 0,9830.$$

Considerando que o dado em ponto flutuante com precisão simples que deve ser transmitido pelos processadores no final de cada cálculo tem 32 *bits*, então

$$\eta_{tx} = 32 / (64 + 48 + 48 + 16 + 360 + 32) \approx 0,0564.$$

Neste caso, embora sejam transmitidos 32 bits o protocolo *Ethernet* envia seu valor mínimo de transmissão, que é 360 *bits*.

A variação apresentada no coeficiente de eficiência de transmissão η_{tx} também contribui para a diferença apresentada na transmissão pelo FTP, uma vez que na transmissão dos 32 bits, a soma dos bits extras, dos protocolos das camadas acima da *Ethernet*, resulta em um valor bem menor que os 12.000 *bits* de capacidade máxima do *frame Ethernet*.

Considerando que, no caso ideal, o programa de transmissão dos dados é compacto, de forma que o computador consiga alimentar a placa com dados suficientes para que ela possa trabalhar a 10 Mbits/seg, então a velocidade efetiva de transmissão de dados V_t para um processador é dada por:

$$V_t = \eta_{tx} * 10 \text{ Mbits/seg} \quad (6.7)$$

Usando a equação (6.7) para a transmissão de 32 bits tem-se:

$$V_t = 0,0564 * 10 \text{ Mbits/seg} = 0,584 \text{ Mbits/seg}.$$

Considerando que a transmissão via FTP, no melhor caso, levou a uma taxa de 0,174 Mbits/seg, pode se ter uma idéia da sobrecarga ocasionada pelos protocolos e serviços da *Internet*.

O tempo t_{tp} necessário para a transmissão dos dados nos quatro processadores é dado por:

$$t_{tp} = (4*32)/V_t \quad (6.8)$$

sendo que, para este caso, é:

$$t_{tp} = (4*32)/(0,584*10^6) = 219,2 \text{ } \mu\text{seg.}$$

Neste caso não é considerado o tempo perdido com prováveis colisões na rede, porque são apenas 4 processadores, e quando um deles está transmitindo um dado os outros estão recebendo (*broadcast*).

Considerando que a arquitetura proposta é constituída de uma rede homogênea de microcomputadores, o tempo de processamento t_p pode ser calculado. Considerando o emprego de processadores 486DX/33, da tabela 5.2 tem-se:

$$t_p = 1/(6.6 * 10^6) * 7 = 1.06 \text{ } \mu\text{seg,}$$

que é o tempo para se realizar as 7 operações em ponto flutuante discutidas na *seção 6.3.2*.

Conclusão sobre a implementação em rede

Considerando que o valor encontrado para o tempo total

de transmissão dos dados de todos os processadores é de 219,2 μ seg e que esses mesmos processadores levam 1,06 μ seg para calcular tais dados, pode-se concluir, com o emprego desta arquitetura e dos processadores 80486DX/33, que o tempo total para a execução com 4 processadores é $220,3/4 = 55,08$ vezes maior que com um único processador.

Com essas velocidades de transmissão, o uso da rede *Ethernet* está definitivamente descartado. Mesmo que fosse empregado o novo padrão *Ethernet* a 100 Mbits/seg [BRYAN, 93], ainda assim um único processador seria 5,5 vezes mais rápido.

6.3.4. Solução via Compartilhamento de Memória

Uma outra forma de implementação da arquitetura paralela, que a exemplo da solução via redes, permite aproveitar a arquitetura aberta dos computadores IBM-PC e da linha de microprocessadores da Intel, é baseada na transmissão de dados via compartilhamento de memória.

Nessa implementação, apenas uma pequena região na parte mais alta da memória (acima dos 16 Mbytes, por exemplo) é compartilhada entre os processadores. Essa abordagem permite utilizar o mesmo sistema operacional e demais programas do sistema utilizados normalmente na linha IBM-PC, ao contrário do que ocorre com multiprocessadores com memória totalmente compartilhada, que requerem um sistema operacional próprio.

Características do sistema

O sistema é implementado de forma que cada processador pode escrever dados em uma palavra de 32 bits (*DW*). Esta palavra pode, posteriormente, ser lida por todos os outros processadores de uma maneira assíncrona, ou seja, um processador pode iniciar a leitura do dado mesmo que outro ainda não tenha terminado seu ciclo de leitura. Com o modo assíncrono de leitura, além do ganho de velocidade, a sincronização dos processadores pode se feita com mais facilidade.

Além da palavra *DW* de 32 bits, cada processador pode escrever em palavras adicionais de 8 bits (*BS* e *CS*), que são utilizadas pelos circuitos para escrever informações de sincronização em registradores de 8 bits (*ST1* e *ST2*), os quais são lidos por todos os processadores.

O sistema é implementado usando computadores tipo IBM-PC, utilizando o barramento de 32 bits padrão EISA. O barramento EISA [GLASS, 89] permite acesso direto de 32 bits na memória, o que é conveniente, pois permite passagem de todo um resultado em ponto flutuante de 32 bits em apenas um ciclo de memória.

Um mapa da memória de cada processador pode ser visto na figura 6.5, onde os primeiros 16 Mbytes são reservados para o sistema operacional e programas do sistema, na forma convencional. Após os 16 Mbytes, estão esquematizadas as 4 palavras de 32 bits (*DW1* ... *DW4*), cada uma composta por 4 bytes. Seguindo estas palavras, estão alocados os "bytes" de sincronização, *BS* e *CS*,

e finalmente os "bytes" de *status*, *ST1* e *ST2*.

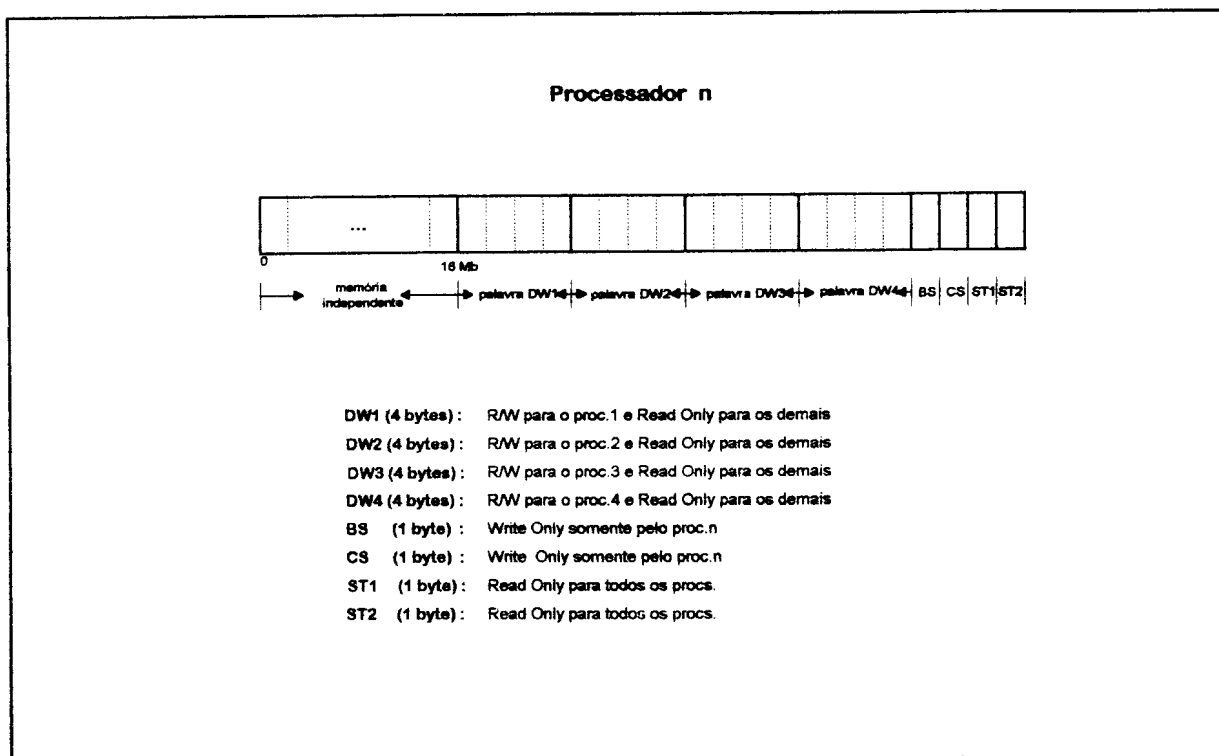


Figura 6.5 - Mapeamento da memória de um processador, com as posições comuns a todos os processadores.

Implementação

Os endereços acima do 16 Mbytes descritos na figura 6.5 são implementados em uma placa de circuito impresso, que é interligada aos barramentos dos processadores através de cabos flexíveis. As palavras de 4 bytes *DW* são implementadas de tal forma que o processador *n* pode ler e escrever em *DW_n*, mas somente ler as palavras *DW_j*, para *j* ≠ *n*.

Cada *bit* do registrador DW é implementado utilizando 1/8 do circuito integrado TTL 74LS273, que possui 8 *flip-flops* do tipo D em um único encapsulamento. As saídas dos *flip-flops* são interligadas aos barramentos de dados dos processadores por chaves de três estados. A figura 6.6 apresenta a esquematização da implementação de um *bit*, dos 32, da palavra DW_n do processador n . Como pode ser observado, apenas o processador n pode passar os dados de seu barramento ao registrador DW_n . O registrador DW_n permite o acesso a um, dois, três ou aos quatro bytes simultaneamente, através das

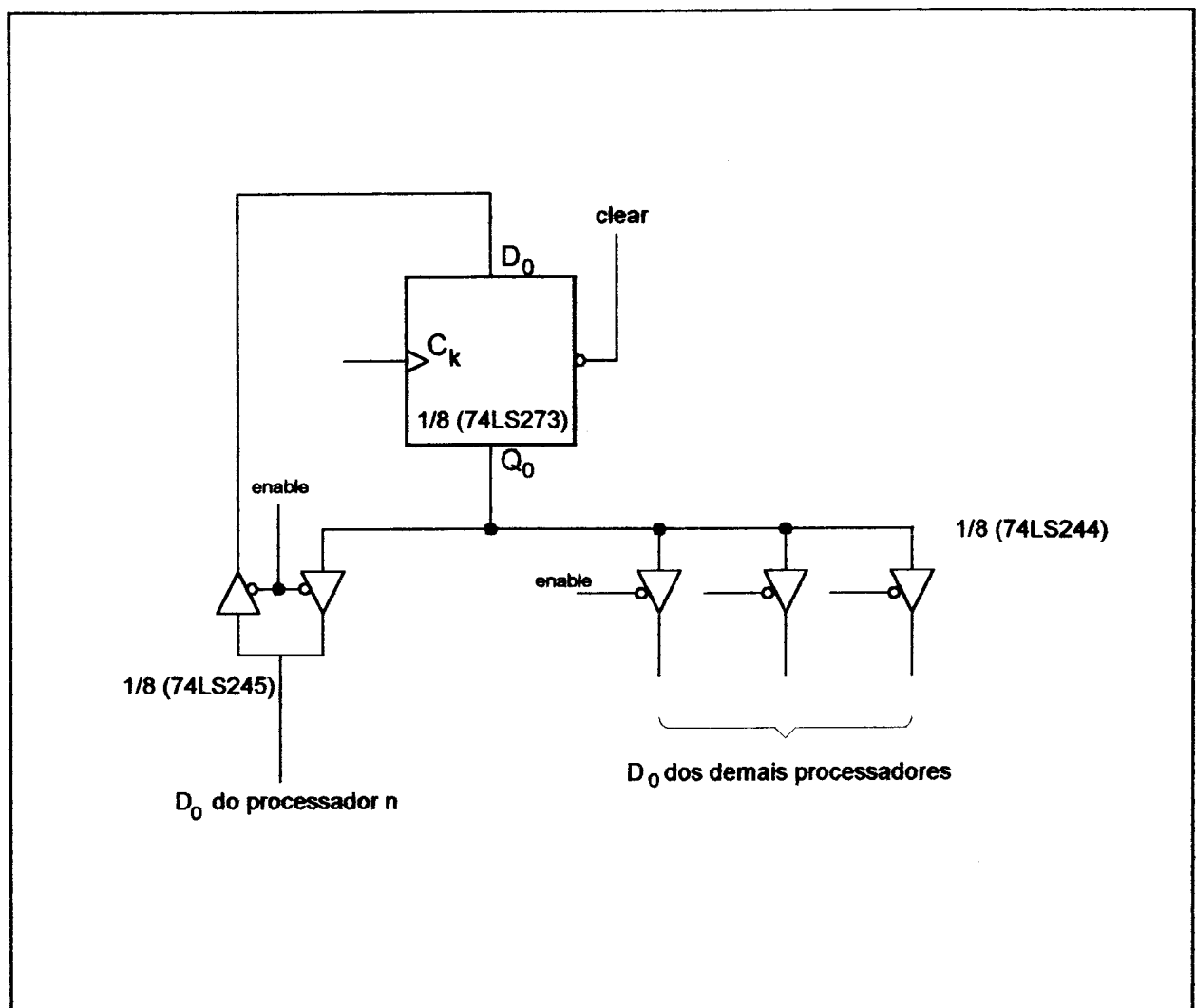


Figura 6.6 - Esquema da implementação de um *bit* da palavra DW_n do processador n .

linhas de *byte enable* $BE0^*$ a $BE3^*$ do barramento EISA (o símbolo $*$ significa que a linha é ativa em zero).

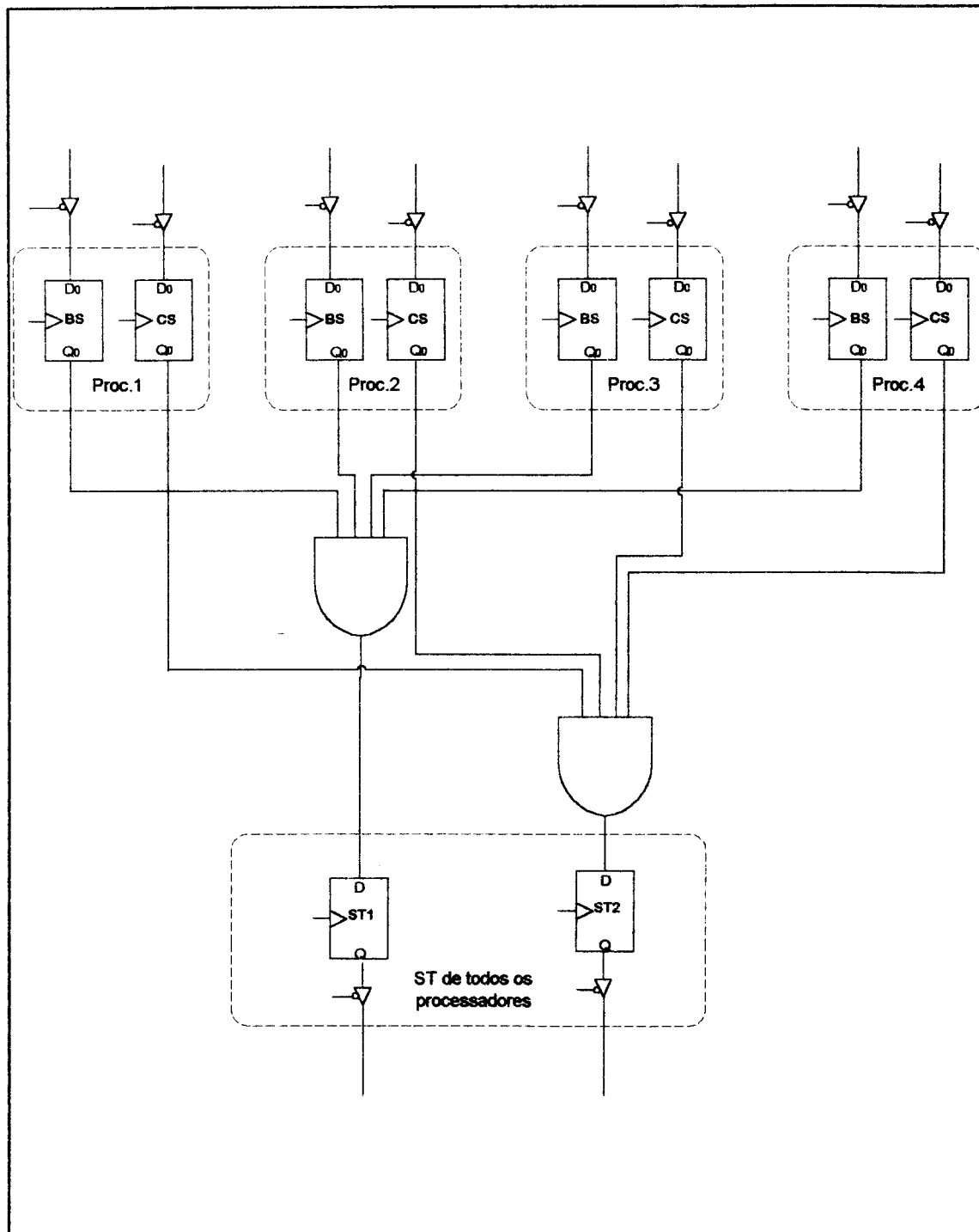


Figura 6.7 - Circuito de sincronismo dos processadores.

As palavras *BS* e *CS* são usadas para sincronização e seus dados são do tipo lógico. Apenas o *bit* menos significativo de *BS* e *CS* são implementados. Cada processador pode apenas escrever nos endereços de *BS* e *CS*, sendo que seus sinais vão para o circuito de sincronismo. A implementação de *BS* e *CS* pode ser vista na figura 6.7.

As palavras *ST1* e *ST2* são lidas por todos os processadores e tem por finalidade fornecer indicações de estado do processamento aos n processadores. Sua implementação também está esquematizada na figura 6.7 e *ST1* e *ST2* também possuem apenas 1 *bit*.

Sincronização

A sincronização do processamento é realizada pelas palavras *BS*, *CS*, *ST1* e *ST2*, como esquematizado na figura 6.7. Estas palavras foram implementadas apenas com um *bit*, para que todas as operações de controle fossem realizadas apenas por instruções de READ e WRITE, evitando operações de mascaramento de *bits* por parte dos processadores. Esse procedimento reduz o número de instruções do programa de sincronização. O algoritmo de sincronização está esquematizado na figura 6.8, cujos passos são descritos a seguir.

No início do processamento, os valores iniciais das variáveis de integração X_1 , X_2 , X_3 e X_4 são armazenados nos registradores DW_1 , DW_2 , DW_3 e DW_4 . Como discutido na seção 6.2.2, cada processador n vai calcular o novo valor de X_n e depois

passá-lo aos demais. Os valores iniciais do *bit* menos significativo de *BS*, *CS*, *ST1* e *ST2*, em todos os processadores, são fixados em 0.

Inicialmente, os quatro processadores realizam os cálculos dos novos valores de X_n . Assim que um processador n termina seus cálculos, ele atribui "1" ao *bit* menos significativo de *BS* em sua memória e vai para uma rotina de espera, onde fica testando o *bit* menos significativo de *ST1*, e só vai sair desta rotina de espera quando este *bit* tiver o valor "1".

Como pode ser observado na figu-

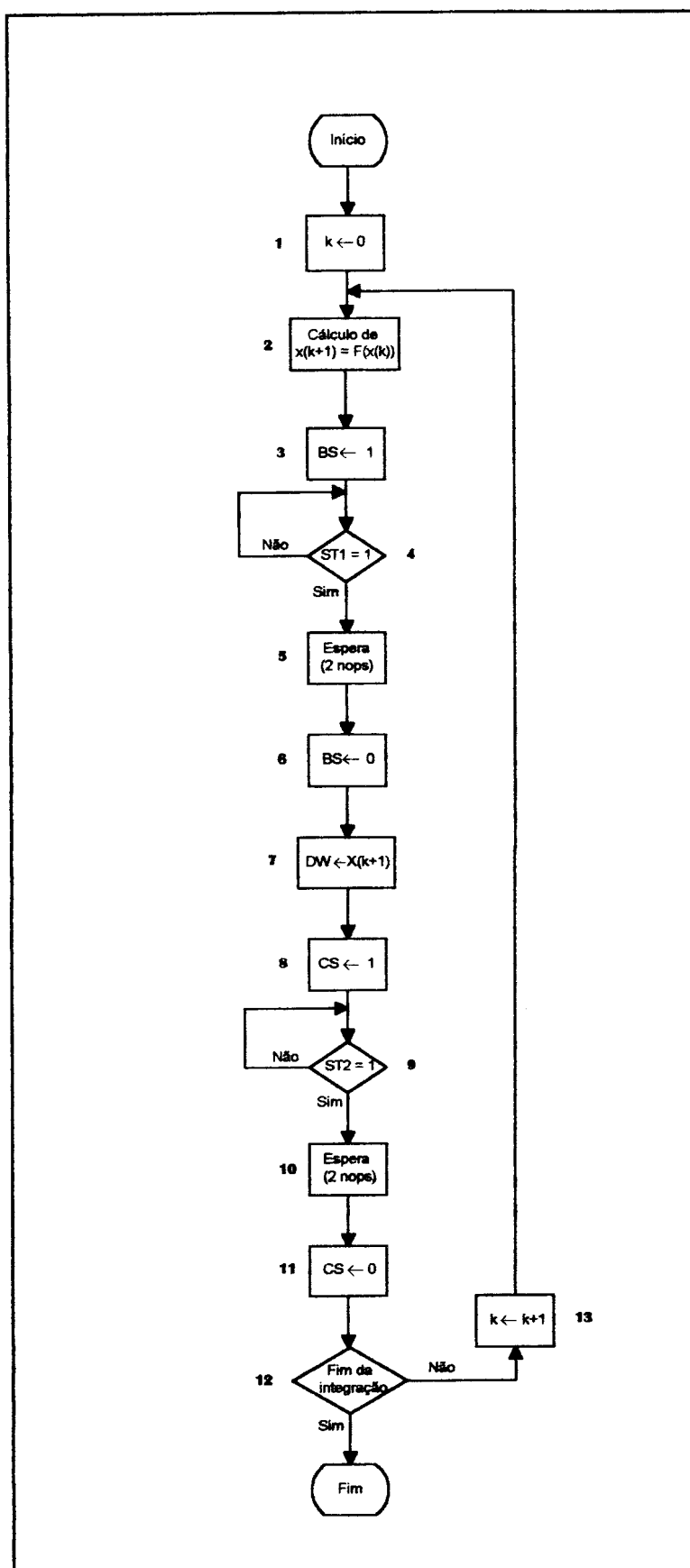


Figura 6.8 - Algoritmo de sincronização dos processadores.

ra 6.7, o *bit* menos significativo de *ST1* vai ser "1" somente quando todos os processadores tiverem terminado seus cálculos. Assim, quando o processador encontrar o valor "1" no *bit* menos significativo de *ST1*, ele sai da rotina de espera, realiza duas instruções de "NOP" e em seguida escreve "0" no *bit* menos significativo de *BS*.

Continuando, o processador *n* escreve o novo valor de X_n em seu registrador *DW*, escreve "1" no *bit* menos significativo do registrador *CS* e vai para outra rotina de espera até que o valor do *bit* menos significativo de *ST2* tenha o valor "1". Quando esta condição é satisfeita, significa que todos os processadores já escreveram seus novos valores de X_n nos registradores *DW_n*. Então, o processador *n* realiza duas instruções de "NOP", escreve "0" no *bit* menos significativo de *ST2* e continua o processo de cálculo do próximo valor de X_n .

Análise de desempenho

Nos casos anteriores, onde foram analisadas as implementações com transputadores e rede de computadores, o tempo de transmissão dos dados era tão longo que, na análise de desempenho dos sistemas, não foi possível levar em conta qualquer tentativa de sincronização do processamento. A sincronização, como aquela discutida nos parágrafos anteriores, garante que, em um determinado instante, todos os processadores trabalhem com os dados referentes a uma mesma iteração.

Na arquitetura com compartilhamento de memória, a medida

do desempenho foi baseada em processadores 80486DX/33. Nos casos em que a velocidade foi determinada pela soma dos ciclos de *clock* de cada instrução, escolheu-se sempre o melhor resultado em termos de velocidade de execução de cada instrução considerada. Isso significa, no caso do 80486, que o *pipeline* foi aproveitado em sua capacidade máxima e que os dados e instruções foram buscados do *cache* interno (não foi considerada penalização por falha na *cache*).

Para efeito de comparação com os casos anteriores, a razão entre tempo de transmissão e tempo de processamento, R_{tp} , pode ser calculada. Nos casos anteriores, foi empregada a velocidade média para operações em ponto flutuante. Já para o processador 80486 esses cálculos podem ser realizados com mais precisão. Considerando, ainda, o caso específico do controle do motor D.C., da *seção 6.2.2*, pode-se observar que a linha que exige cálculos mais complexos requer uma adição, dois produtos e uma divisão. Para a realização dessas operações em ponto flutuante de 32 bits, considerando o melhor resultado, como mencionado anteriormente, são necessários 112 ciclos de *clock*, já incluindo os ciclos de *load* (carregamento) dos dados nos registradores. A transmissão deste registrador para a posição de memória *DW* é realizada em um único ciclo de *clock*. No pior caso, este dado deve ser transferido para uma outra posição de memória, e depois passado para *DW*. Nessa situação, seriam gastos 3 ciclos de *clock*, sendo que o ciclo necessário para o armazenamento na posição intermediária de memória seria considerado parte do tempo de processamento. Assim, segundo a equação (6.1),

$$R_{cp} = 2/112 \approx 0,018.$$

Dessa forma, o coeficiente de rendimento pela equação (6.2) é:

$$\eta = 1/(1 + 0,018) \approx 0,982.$$

Assim sendo, o ganho de velocidade para 4 processadores, segundo a equação (6.3), é:

$$G_p = \eta * 4 \approx 3,928.$$

Portanto, das três formas de implementação, esta é a que apresenta melhores resultados. Entretanto, como já foi visto anteriormente, sendo obtida a velocidade para a transferência de dados, é necessário também levar em conta o tempo gasto com a sincronização dos processadores. A sincronização evita, por exemplo, que em um dado instante do tempo, os processadores trabalhem com versões diferentes dos dados. Por exemplo, quando um processador está calculando o novo valor de X após k iterações, todos os outros devem estar nesta mesma fase de cálculo.

Para a determinação do tempo de sincronização, considerou-se que, na execução em paralelo, cada linha em **negrito** na **seção 6.2.2** fosse atribuída a um processador. Se for considerada a linha que necessita de maior tempo de execução, o processador correspondente a esta linha não chegará a entrar em *loop* de espera na rotina de sincronização, pois ele será o último a terminar. Esse procedimento permite se conhecer exatamente o número de instruções

que serão executadas dentro do laço de espera, permitindo que o tempo de execução da rotina possa ser determinado.

O tempo de execução da rotina de sincronização, por sua vez, pode ser obtido por uma simulação simplificada, onde um único processador realiza a tarefa que lhe caberia se estivesse trabalhando em paralelo com os outros. Como já visto no parágrafo anterior, se for escolhida, para este processador, a linha com os cálculos mais demorados, ele não entrará na rotina de espera, e seu tempo de execução poderá, então, ser determinado de uma das duas maneiras abaixo:

- (1) por medida real do tempo de execução, onde um *clock* é lido no início do programa e outro no término do mesmo;
- (2) por soma do tempo de execução de cada instrução em um programa em linguagem de máquina.

Seguindo o modo (1) acima, utilizou-se um procedimento em FORTRAN 5.0 da *Microsoft* para calcular o novo valor de $X(k+1)$ (bloco 2 no algoritmo da figura 6.8). Uma vez que o FORTRAN não possui funções para armazenar e retirar valores em endereços específicos da memória, os valores das variáveis *BS* e *CS* foram passados como parâmetros para um procedimento em Assembly, a fim de armazenar esses valores nas correspondentes posições de memória. Da mesma forma, a leitura das variáveis de *status* *ST1* e *ST2* e os respectivos procedimentos de espera (blocos 3,4 e 8,9) foram implementados em procedimentos em Assembly, também chamados pelo procedimento FORTRAN mencionado anteriormente. O tempo de computação foi determinado através de leituras do *clock* do sistema, no início e

no fim do procedimento em FORTRAN.

Na rotina de simulação todo o procedimento foi repetido 100.000 vezes para se obter maior precisão na medida do tempo, pois o relógio (*clock*) do sistema tem uma precisão de centésimo de segundo. Assim, para este ciclo de 100.000 vezes, o cálculo de $X(k+1)$, em FORTRAN, gastou 49 centésimos de segundos, sendo que o tempo total de execução, incluindo a rotina de sincronização, foi de 110 centésimos de segundos. Destes 110 centésimos, cada vez que o programa FORTRAN chamava uma rotina em Assembly e passava um número de 32 *bits* eram gastos 12 centésimos de segundos.

Uma vez que o tempo para a passagem de parâmetros entre a rotina em FORTRAN e as rotinas em Assembly é significativo, o programa de sincronização foi modificado, de forma que o programa em FORTRAN realiza apenas uma chamada da rotina em Assembly. Nesse caso, o tempo total de execução (cálculo de $X(k+1)$ e rotina de sincronização) foi reduzido para 77 centésimos de segundos. Portanto, a razão entre o tempo total de processamento e o tempo de sincronização é $49/77 \approx 0,636$.

É interessante observar que, nesse caso, o tempo gasto na rotina de sincronização é equivalente ao tempo de transmissão dos resultados nos casos anteriores, e a razão calculada no parágrafo anterior é justamente o coeficiente de rendimento. Isto porque o coeficiente de rendimento é uma razão entre o tempo de processamento útil (processamento de $X(k+1)$) e o tempo total (processamento de $X(k+1)$ e sincronização) - o tempo de transmissão é realizado em apenas 1 ciclo de *clock* e já está incluído na

rotina de sincronização. Assim, para o caso da interação do FORTRAN com o Assembly,

$$\eta = 0,636$$

e o ganho de velocidade em relação um único processador é

$$G_p = \eta * 4 \approx 2,54.$$

O coeficiente de rendimento pode ser aumentado se todos os cálculos forem realizados em linguagem Assembly. Nesse caso, como já visto anteriormente, são necessários 112 ciclos de *clock* para a realização do cálculo de $X(k+1)$ (uma adição, dois produtos, uma divisão e mais o armazenamento na memória). A parte do código referente ao algoritmo de sincronização requer 26 períodos de *clock*, quando não é necessário entrar em *loops* de espera. Portanto, no melhor dos casos, o rendimento é

$$\eta = 112 / (112 + 26) \approx 0,812$$

e o ganho de velocidade para quatro processadores é

$$G_p \approx 3,25.$$

Conclusão

Os ganhos de velocidade obtidos na seção anterior permitem concluir que esta implementação da arquitetura é bastante

viável. Ela apresenta um esquema relativamente simples de sincronização dos processadores e ainda permite que sejam utilizados sistemas de microcomputadores baseados em arquitetura aberta "de fato" (IBM-PC).

O uso dos sistemas baseados na linha de microcomputadores compatíveis com o IBM-PC mantém a arquitetura implementada sempre atual, pois basta utilizar os processadores "topo de linha" dessa família. Isso porque a transmissão dos dados entre os processadores é realizada em um único ciclo de *clock*. Mesmo com a troca do microprocessador, o ganho de velocidade G_p não se altera significativamente. Dessa forma, se forem utilizados 4 processadores *Pentium*, o desempenho total será cerca de 3,25 vezes o desempenho de um deles.

Outra vantagem desta implementação é que os programas podem ser desenvolvidos em FORTRAN - uma linguagem simples, ainda muito utilizada em processamento numérico; apenas a rotina discutida na seção anterior tem que ser implementada em Assembly. Em uma implementação com transputadores, possivelmente, todos os programas deveriam ser implementados em Occam, que não expressa a "legibilidade" e "escribabilidade" de Fortran, além de não ser muito difundida em computação numérica.

CAPÍTULO 7

Conclusão

O método do Gradiente Conjugado é uma ferramenta eficiente para a solução de problemas de controle ótimo. O algoritmo do gradiente conjugado é relativamente simples, e permite a obtenção das entradas de controle ótimo independente da linearidade do sistema e do funcional de custo. A extensão do método para problemas de controle ótimo preserva suas características de estabilidade e convergência rápida. Além disso, ele permite a solução mesmo quando as variáveis de estado e de controle estão sujeitas a restrições de saturação.

O método é bastante geral e pode ser aplicado na obtenção de outros tipos de entradas de controle, diferentes do caso de funções contínuas considerado neste trabalho. O gradiente conjugado pode ser aplicado com sucesso na obtenção de entradas de controle para outros espaços de domínio, como o espaço de modulação por largura de pulsos, conjunto de parâmetros e entradas amostradas. Os problemas típicos que podem ser resolvidos

nesses espaços são: controle de posição e temperatura por modulação em largura de pulso; determinação de parâmetros em compensadores, processos químicos e processos em geral; controle por entradas amostradas; entre outros [HASDORFF, 76][FISCHER, 83].

A maior limitação do método do gradiente conjugado é que o mesmo deve ser calculado em um intervalo de operação $\Delta t_{op} = [t_0, t_f]$, onde este intervalo deve ser determinado antes do início dos cálculos. Tal limitação impedia a aplicação em problemas de controle ótimo em tempo real. Entretanto, os algoritmos desenvolvidos neste trabalho possibilitam tal aplicação.

O insucesso na tentativa de se realizar o controle ótimo em tempo real, ampliando o intervalo de tempo real, Δt_{op} , para um intervalo muito grande (*capítulo 4*) levou à formulação da *Conjectura 4.1*, a qual permitiu a proposição de um algoritmo para o controle ótimo com o tempo final livre. O excelente desempenho desse algoritmo, em termos de resposta do sistema (*figura 4.11*), permitiu que o mesmo fosse generalizado para a solução de problemas em tempo real.

A generalização apresentada permite a aplicação do controle ótimo em tempo real, tanto em malha aberta como em malha fechada. O caso de malha fechada é interessante, porque permite que as variáveis do sistema sigam as variáveis do modelo, independente das perturbações aplicadas ao sistema (*figura 4.21*).

A simulação do processo sujeito a perturbações, realizada no *capítulo 4*, mostrou que, do ponto de vista do siste-

ma, não há diferença entre as abordagens com restrições severas de tempo de processamento e a do algoritmo de canalização (*pipeline*), onde é branda a restrição no tempo de processamento. Este fato abre uma excelente perspectiva para a implementação do controle ótimo em tempo real, utilizando os microprocessadores atuais.

A análise de desempenho dos microprocessadores (*capítulo 5*) revelou que a família 80x86, principalmente a partir do 80486 de 33 Mhz, é uma excelente alternativa para a implementação do controle ótimo em tempo real, mesmo quando comparada a processadores como os transputadores (*transputers*) T800 e T9000. Este é um resultado interessante, quando consideramos que os computadores tipo PC-AT, baseados na família 80x86, são praticamente sistemas de desenvolvimento abertos "de fato", enquanto que o uso de transputadores envolve ou a compra ou o desenvolvimento de um sistema proprietário, tanto a nível de circuitos como de programas.

Para obter maior velocidade de processamento, explorando o paralelismo inerente do método do gradiente conjugado, foram analisadas três arquiteturas paralelas para aplicações em tempo real (*capítulo 6*): uma com o uso de 4 transputadores T800, uma utilizando rede de computadores e outra com microcomputadores compartilhando trechos comuns de memória.

A análise realizada mostrou que a implementação da arquitetura paralela com transputadores apresenta um desempenho inferior ao obtido com um único processador 80486DX/33. Mesmo se

os T800 fossem substituídos pelo novo transputador T9000, o desempenho ainda ficaria próximo ao de um único *Pentium* P54C.

Também a implementação via rede de computadores não apresentou eficiência, sendo ainda mais lenta que os transputadores.

Já a implementação usando microcomputadores compatíveis com o IBM-PC, que compartilham algumas posições de memória, apresentou resultados satisfatórios, chegando a atingir cerca de 3,25 vezes a velocidade de um único processador, e ainda assim permitir um mecanismo eficiente de sincronização entre os processadores. Essa implementação tem, ainda, a vantagem de manter aproximadamente o mesmo ganho de velocidade, mesmo para os processadores "topo de linha" da família IBM-PC.

Referências Bibliográficas

- [ABADIE, 78] ABADIE, J. **Non Linear and Integer Programming.** North-Holland Publishing, 1978, capítulo 8
- [ALPERT, 93] ALPERT, D. & AVNON, D. **Architecture of the Pentium Microprocessor.** IEEE Micro, pp.11-21, June 1993
- [AOKI, 71] AOKI, M. **Introduction to Optimization Techniques: Fundamentals and Application of Nonlinear Programming.** The Macmillan Company, New York, USA, 1971
- [ÅSTRÖM, 70] ÅSTRÖM, K.J. **Introduction to Stochastic Control Theory.** Academic Press, California, USA, 1970, 299p.
- [ÅSTRÖM, 89] ÅSTRÖM, K.J. & WITTENM, B. **Adaptive Control.** Addison-Wesley Publishing Company, USA & Canada, 1989, 526p.
- [ATHANS, 66] ATHANS, M. & FALB, P.L. **Optimal Control.** McGraw-Hill, Inc., USA, 1966, 879p.
- [BELLMAN, 62] BELLMAN, R. & DREYFUS, S.E. **Applied Dynamic Programming.** Princeton University Press, New Jersey, USA, 1962, 363p.
- [BELLMAN, 64] BELLMAN, R. & KALABA, R. **Selected Papers on Mathematical Trends in Control Theory.** Dover Publications, Inc., New York, USA, 1964, 200p.

- [BELLMAN, 65] BELLMAN, R. & KALABA, R. **Dynamic Programming and Modern Control Theory.** Academic Press, New York, USA, 1965, 111p.
- [BERTSEKAS, 74] BERTSEKAS, D.P. **Partial Conjugate Gradient Methods for a Class of Optimal Control Problems.** IEEE Transactions on Automatic Control, AC-19, N.3, pp.209-217, June 1974
- [BENNETT, 90] BENNETT, S. & VIRK, G.S. **Computer Control of Real-Time Processes.** Peter Peregrinus Ltd., London, England, 1990, 306p.
- [BERTSEKAS, 76] BERTSEKAS, D.P. **Dynamic Programming and Stochastic Control.** Academic Press, New York, USA, 1976, 397p.
- [BOTTURA, 77] BOTTURA, C.P.; MONTICELLI, A.; AQUINO, L.A.C. **Controle Digital Ótimo de Processo Eletromecânico: Projeto e Implementação de Algoritmo em Computador Híbrido.** Anais: 1º Congresso Brasileiro de Automática, São Paulo-SP, Brasil, 1976
- [BOX, 66] BOX, M.J. **A Comparison of Several Current Optimization Methods and the Use of Transformation in Constrained Problems.** The Computer Journal, V.9, pp.67-77, May 1966
- [BOX, 76] BOX, G.E. & JENKINS, G.M. **Time Series Analysis - Forecasting and Control.** Holden-Day, Inc., California, 1976, 575p.
- [BRYAN, 93] BRYAN, J. **Pumping Up Ethernet.** Byte, V.18, N.9, pp.121-126, August 1993

- [CADZOW, 70] CADZOW, J.A. & MARTENS, H.R. **Discrete-Time and Computer Control Systems.** Prentice-Hall, Inc., New Jersey, USA, 1970, 473p.
- [CHEN, 70] CHEN, C.T. **Introduction to Linear System Theory.** Holt, Rinehart & Winston, Inc., USA, 1970, 431p.
- [CLULEY, 75] CLULEY, J.C. **Computer Interfacing and On-Line Operation.** Crane, Russak & Company, Inc., New York, USA, 1975, 181p.
- [COMMER, 91] COMMER, D.E. **Internetworking with TCP/IP.** Prentice-Hall International Inc., USA, 1991, 547p.
- [CRAIG, 88] CRAIG, J.J. **Adaptive Control of Mechanical Manipulators.** Addison-Wesley Publishing Company, USA & Canada, 1988, 136p.
- [CRONE, 93] CRONE, L.G.C. & VAN DER VORST, H.A. **Communication Aspects of the Conjugate Gradient Method on Distributed-Memory Machines.** Supercomputer, N.58, pp.4-9, 1993
- [CRONE, 94] CRONE, L.G.C. & VAN DER VORST, H.A. **Communication Aspects of the Conjugate Gradient Method on Distributed-Memory Machines.** Versão revisada de janeiro de 1994. Separata
- [CURNOW, 76] CURNOW, H.J. & WICHMANN, B.A. **A Synthetic Benchmark.** Computer Journal, V.19, N.1, pp.43-49, 1976
- [DAVIDON, 59] DAVIDON, W.C. **Variable Metric Method for Minimization.** A.E.C. Research and Development Report, ANL-5990 (Rev.), 1959

- [DAVIS, 85] DAVIS, M.H.A. & WINTER, R.B. **Stochastic Modelling and Control**. Chapman & Hall, London, United Kingdom, 1985, 393p.
- [D'AZZO, 75] D'AZZO, J.J. & HOUPIS, C.H. **Linear Control System Analysis and Design**. McGraw-Hill, Inc., USA, 1975, 636p.
- [DIGITAL, 91] DIGITAL EQUIPEMENT CORPORATION **Open Systems Handbook. A Guide to Building Open Systems**. Digital 1991
- [DUNCAN, 90] DUNCAN, R. **A Survey of Parallel Computer Architectures**. IEEE Computer, February 1990
- [EVANS, 93] EVANS, D.J. & GALLIGANI, E. **A Parallel Additive Preconditioner for Conjugate Gradient Method for $AX + XB = C$** . Parallel Computing, N.20, pp.1055-1064, 1994
- [FISCHER, 83] FISCHER, B.R. **Análise e Implementação de Algoritmos de Controle Ótimo pelo Método do Gradiente Conjugado**. Dissertação de Tese de Mestrado: Faculdade de Engenharia, Unicamp, Campinas-SP, Brasil, 1983, 170p.
- [FLETCHER, 63] FLETCHER, R. & POWELL, M.J.D. **A Rapidly Convergent Descent Method for Minimization**. The Computer Journal, V.6, pp.163-168, 1963
- [FLETCHER, 64] FLETCHER, R. & REEVES, C.M. **Function Minimization by Conjugate Gradients**. The Computer Journal, V.7, N.7, pp.149-153, July 1964
- [FLETCHER, 69] FLETCHER, R. **Optimization : "A Review of Methods for Unconstrained Optimization"**. Academic Press, New York & London, 1969

- [FLETCHER, 93] FLETCHER, P. & NASS R. **Transputer Claims Single-Chip Speed Record.** Eleconic Design, May 13, pp.95-100, 1993
- [FONG, 71] FONG, T.S. & LEONDS, C.T. **Method of Conjugate Gradients for Optimal Control Problems with State Variable Constraint - Advances in Control Systems.** Academic Press, New York, USA, Vol.8, 1971
- [FRIED, 91] FRIED, S.S. **Personal Supercomputing with the Intel i860.** Byte, V.16, N.1, pp.347-358, January 1991
- [GEE, 93] GEE, J.D. et al. **Cache Performance of the SPEC92 Benchmark Suite.** IEEE micro, pp. 17-58, August 1993
- [GLASS, 89] GLASS, L.B. **Inside EISA.** Byte, V.14, N.12, pp.417-425, 1989
- [HALFHILL, 93] HALFHILL, T.R. **Intel Launches Rocket in a Socket.** Byte, V.18, N.6, pp.92-108, 1993
- [HALFHILL, 94a] HALFHILL, T.R. **80x86 Wars.** Byte, V.19, N.6, pp.74-88, 1994
- [HALFHILL, 94b] HALFHILL, T.R. **AMD vs. Superman.** Byte, V.19, N.11, pp.95-104, 1994
- [HASDORFF, 76] HASDORFF, L. **Gradient Optimization and Nonlinear Control.** John Wiley & Sons, New York, USA, 1976
- [HATLEY, 91] HATLEY, D.J. & PIRBHAI, I.A. **Estratégias para Especificação de Sistemas em Tempo Real.** Makron Books do Brasil Ltda., São Paulo, Brasil, 1991, 455p.

- [HAYES, 89] HAYES, F. **Battle of the Chips.** Byte, V.14, N.3, pp.274-279, March 1989
- [HESTENES, 52] HESTENES, M.R. & STIEFEL, E. **Methods of Conjugate Gradients for Solving Linear Systems.** J.Res.Nat.Bur.Standards, V.49, pp.409-436, 1952
- [HOCKNEY, 88] HOCKNEY, R.W. & JESSHOPE C.R. **Parallel Computers 2.** Adam Hilger, Bristol UK, 2nd ed., 1988 623p.
- [HOUPIS, 85] HOUPIS, C.H. & LAMONT, G.B. **Digital Control Systems.** McGraw-Hill Book Company, Singapore, 1985, 667p.
- [INMOS, 89] INMOS Limited **Transputer Data Book.** 1989
- [INTEL, 90] INTEL THE MICROCOMPUTER Company **Microprocessors.** Order Number 230843, Intel 1990
- [JACOBSON, 69] JACOBSON, D.H. & LELE, M.M. **A Transformation Technique for Optimal Control Problems with a State Variable Inequality Constraint.** IEEE Transactions on Automatic Control, V.AC-14, N.5, pp.457-464, October 1969
- [JONES, 88] JONES, G. & GOLDSMITH, M. **Programming in Occam 2.** Prentice-Hall International, United Kingdom, 1988, 317p.
- [LANDAU, 79] LANDAU, Y.D. **Adaptive Control.** Marcell Dekker Inc., New York & Basel, 1979, 406p.
- [LASDON, 67a] LASDON, L.S.; WARREN, A.D.; RICE, R.K. **An Interior Penalty Method for Inequality Constrained Optimal Control Problems.** IEEE Transactions on Automatic Control, AC-12, N.4, pp.387-394, August 1967.

- [LASDON, 67b] LASDON, L.S.; MITTER, S.K.; WARREN, A.D. **The Conjugate Gradient Method for Optimal Control Problems.** IEEE Transactions on Automatic Control, AC.12, N.2, pp.132-138, April 1967
- [LASDON, 70] LASDON, L.S. **Conjugate Direction Methods for Optimal Control.** AC-15, N.2, pp.267-268, April 1970
- [LUEMBERGER, 69] LUEMBERGER, D.G. **Optimization by Vector Space Methods.** John Wiley & Sons, Inc., New York, USA, 1969, 326p.
- [LUEMBERGER, 73] LUEMBERGER, D.G. **Introduction to Linear and Nonlinear Programming.** Addison-Wesley, 1973
- [MAGALHÃES, 90] MAGALHÃES, M.F. **Sistemas de Tempo Real.** Minicursos: 8º Congresso Brasileiro de Automática, Belém-PA, Brasil, pp.78-97, 1990
- [MARGULIS, 89] MARGULIS, N. **The Intel 80860.** Byte, V.14, N.13, pp.333-340, December 1989
- [MIELE, 79] MIELE, A.; WU, A.K.; LIU, C.T. **A Transformation Technique for Optimal Control Problems with Partially Linear State Inequality Constraints.** Journal of Optimization Theory and Applications, V.28, N.2, pp.185-212, June 1979
- [NADEAU, 89] NADEAU, M.E. **Battle of the SXs.** Byte, V.14, N.3, pp. 278-279, March 1989
- [NETUSHIL, 73] NETUSHIL, A. *General Ed.* **Theory of Automatic Control.** Mir Publishers, Moscow, URSS, 1973, 806p.
- [NOTON, 72] NOTON, M. **Modern Control Engineering.** Pergamon Press, Inc., New York, 1972

- [PAGUREK, 68] PAGUREK, B. & WOODSIDE, C.M. **The Conjugate Gradient Method for Optimal Control Problems with Bounded Control Variables.** Automatica, V.4, pp.337-349, 1968
- [PANDIT, 83] PANDIT, M.S. & WU, S.M. **Time Series and System Analysis with Applications.** John Wiley & Sons, Inc., New York, USA, 1983, 586p.
- [POUNTAIN, 91] POUNTAIN, D. **The Transputer Strikes Back.** Byte, V.16, N.8, pp.265-275, August 1991
- [POWELL, 64] POWELL, M.J.D. **An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives.** The Computer Journal, V.7, N.2, pp.155-162, Jul.1964
- [POWELL, 70] POWELL, M.J.D. **A Survey of Numerical Methods for Unconstrained Optimization.** SIAM Review, V.12, N.1, pp. 79-97, January 1970
- [PRESSMAN, 87] PRESSMAN, R.S. Software Engeneering **A Practitioner's Approach.** 2nd ed., McGraw-Hill International Editions, Singapore, 1987, 567p.
- [PUCCINI, 80] PUCCINI, A.L. **Introdução à Programação Linear.** Livros Técnicos e Científicos Ed., Rio de Janeiro, Brasil, 1980
- [RALSTON, 76] RALSTON, A. & WILF, H.S. **Mathematical Methods for Digital Computers.** John Wiley & Sons, Inc., New York, USA, 1976
- [ROSCH, 90] ROSCH, W.L. **Desvendando o Hardware do PC.** Editora Campus Ltda., Rio de Janeiro, BR, 1990, 522p.

- [ROSEMBROCK, 70] ROSEMBROCK, H.H. **An Automatic Method for Finding the Greatest or Least Value of a Function.** The Computer Journal, V.3, N.3, pp.175-184, October 1970
- [RYAN, 90] RYAN, B. **A Marriage Made in Silicon.** Byte, vol. 15, N.1, pp.261-266, january, 1990
- [RYAN, 93] RYAN, B. **Inside the Pentium.** Byte, V.18, N.6, pp.102-104, May 1993
- [RYAN, 94] RYAN, B. **Alpha Rides High.** Byte, V.19, N.10, pp.197-198, October 1994
- [SAGE, 68] SAGE, A.P. **Optimum Systems Control.** Prentice-Hall, Inc., New Jersey, USA, 1968
- [SAKAROVITCH, 70] SAKAROVITCH, M. **Notes on Linear Programming.** Van Nostrand Reinhold, New York, USA, 1970, 175p.
- [SMARTE, 90] SMARTE, G. **15 Years of Bits, Bytes, and Other Great Moments.** Byte, V.15, N.9, pp.369-400, November 1990
- [STANKOVIC, 88] STANKOVIC, J.A. **A Serious Problem for Next-Generation Systems.** IEEE Computer, pp.10-19, 1988
- [STEIN, 88] STEIN, R.M. **T800 and Counting.** Byte, V.13, N.12, pp.287-296, November 1988
- [TABAK, 71] TABAK, D. & KUO, B.C. **Optimal Control by Mathematical Programming.** Prentice-Hall, Inc., New Jersey, USA, 1971
- [TABAK, 90] TABAK, D. **Risc Systems.** Research Studies Press Ltd., England, 1990, 300p.

- [TANENBAUM, 88] TANENBAUM, A.S. **Computer Networks**. Prentice-Hall International, Inc. , 2nd ed., Singapore, 1988, 658p.
- [TREIMAN, 1990] TREIMAN, J.S. **Optimal Control with Small Generalized Gradients**. SIAM J. Control and Optimizations, V.28, N.3, pp.720-732, May 1990
- [TRIPATHI, 70] TRIPATHI, S.S. & NARENDRA, K.S. **Optimization Using Conjugate Gradient Methods**. IEEE Transactions on Automatic Control, AC-3, N.1, pp.268-270, April 1970
- [WEICKER, 84] WEICKER, R.P. **Dhrystone: A Synthetic Systems Programming Benchmark**. Comm. ACM, V.27, N.10, pp. 1013-1030, October 1984
- [WESTON, 94] WESTON, R. **Os Resultados Podem Variar**. Byte Brasil, V.3, N.11, pp. 116-123, Novembro 1994
- [WHITE, 73] WHITE, W.W. **A Status Report on Computing Algorithms for Mathematical Programming**. Computing Surveys, V.5, N3, pp.135-166, September 1973
- [WIDROW, 85] WIDROW, B. & STEARNS, S.D. **Adaptive Signal Processing**. Prentice-Hall, Inc., New Jersey, USA, 1985, 474p.
- [WISMER, 78] WISMER, D.A. & CHATTERGY, R. **Introduction to Nonlinear Optimization: A Problem Solving Approach**. North-Holland Series in Systems Science and Engeneering, 1978
- [ZANGWILL, 67] ZANGWILL, W.I. **Minimizing a Function without Calculating Derivatives**. The Computer Journal, V.10, pp.293-296, November 1967

[ZOUTENDIJK, 76] ZOUTENDIJK, G. **Mathematical Programming Methods.**
North-Holland Publishing Comp., New York, USA,
1976

APÊNDICE A

ESPAÇO DE HILBERT

Este Apêndice não visa definir toda a teoria envolvendo o espaço de *Hilbert*, mas apenas lembrar e reforçar a definição adotada neste trabalho. Para isso usou-se como referência, as definições apresentadas por Hasdorff em [HASDORFF, 76], dadas a seguir.

Definição de um ESPAÇO LINEAR:

Diz-se que um espaço Γ é linear se

$$\forall x_1, x_2 \in \Gamma \rightarrow (ax_1 + bx_2) \in \Gamma,$$

onde a e b são constantes escalares.

Exemplos: espaço dos números reais (\mathbf{R}^1), espaço euclidiano n -dimensional, real (\mathbf{R}^n) e o espaço de funções contínuas sobre um intervalo real $[a, b]$ ($\mathbf{C}_{[a,b]}$).

Definição de PRODUTO ESCALAR sobre um ESPAÇO:

Um produto escalar sobre um espaço Γ é uma operação que, para todo par de elementos x, y em Γ , atribui um número real, denotado por $\langle x, y \rangle$, com as seguintes propriedades:

1. $\langle x, y \rangle \in \mathbb{R}^1$
2. $\langle x, y \rangle = \langle y, x \rangle$
3. $\langle ax + bz, y \rangle = a\langle x, y \rangle + b\langle z, y \rangle$
4. $\langle x, x \rangle > 0$ se $x \neq 0$
5. $\langle 0, x \rangle = 0$, onde 0 é o elemento nulo.

Definição de um PONTO DE ACUMULAÇÃO:

Diz-se que um ponto ou elemento y é um ponto de acumulação (*cluster point*) de uma sequência $\{x_n\}$ se toda vizinhança ϵ de y contiver um número infinito de membros da sequência, ou seja, $\forall \epsilon > 0$ há um número infinito de x_n tal que:

$$\|y - x_n\| < \epsilon$$

onde a norma de um elemento x é a mesma da definição usual: a raiz quadrada do produto escalar $\langle x, x \rangle$.

Definição de um ESPAÇO COMPLETO:

Diz-se que um espaço Γ é completo se toda sequência nesse espaço, com ponto de acumulação y , tem que $y \in \Gamma$.

Definição do ESPAÇO de HILBERT:

Diz-se que um espaço é de Hilbert quando o espaço possui produto escalar, é linear e completo.

Exemplos: \mathbb{R}^1 , \mathbb{R}^n e $\mathbb{C}_{[a,b]}$

APÊNDICE B1

LISTAGEM DO SUBPROGRAMA DE MINIMIZAÇÃO PELO MÉTODO DO GRADIENTE CONJUGADO

- PARTE INDEPENDENTE DA APLICAÇÃO -

O algoritmo a seguir implementa as subrotinas independentes do problema a ser resolvido pelo método do gradiente conjugado. Normalmente, essas rotinas podem ser aplicadas em entradas de controle no espaço de funções contínuas, entradas amostradas, conjunto de parâmetros, modulação por largura de pulso e em um espaço genérico definido pelo usuário. O programa implementa, ainda, o algoritmo de Pagurek [PAGUREK,68], para o tratamento de restrições de saturação nas variáveis de estado e de controle.

```

C          GRAMIN = MINIMIZAÇÃO PELO GRADIENTE CONJUGADO
C
SUBROUTINE GRAMI
  DIMENSION ALF0(201)
  COMMON U(201),G(201),X(10,201),ALBDA(10,201)
  COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
  COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
C
  CALL INITI
  CALL GRAD
  NP=NPTOSG+1
  NPT=NPTOS+1
  DO 10 I=1,NP
10    SI1(I)=-1.*G(I)
  DO 20 I=1,NITER
20    ALF0(I)=0
    NALF0=1.
30    CALL BUSCA(ALFA,ALF0,NALF0)
    ALF0(NALF0)=ALFA
    DO 40 I=1,NP
40    U(I)=U(I)+ALFA*SI1(I)
    CALL ESCSU(G,G,MODE,ESC,NPTOSG,DT)
    ESC1=ESC
    CALL GRAD
    CALL PAGI(CT1,G,NP)
    CALL ESCSU(CT1,CT1,MODE,ESC,NPTOSG,DT)
    CALL CUSTO(U,ACPRT)
    IF(ESC1)81,80,81
80    ESC1=1E-30
81    BETA=ESC/ESC1
    DO 50 I=1,NP
50    SI1(I)=BETA*SI1(I)-G(I)
    NALF0=NALF0+1
379  FORMAT(2X,5G20.8)
378  FORMAT(2X,6G20.8)
    IF(NITER-NALF0)60,30,30
60  RETURN
    END
C
SUBROUTINE RUNGE(T,H,XOUL,UDT)
  DIMENSION F(11),UDT(201),Y(12),AK(12)
  COMMON U(201),G(201),X(10,201),ALBDA(10,201)
  COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
  COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
  COMMON A(4),B(4),C(4),N11
  EQUIVALENCE(Y(2),X1(1)),(AK(2),F(1))
  Y(1)=T
  AK(1)=1.
  DO 10 J=1,4
  IF(XOUL)7,7,8
C      XOUL=0 PARA FUNX
C      XOUL=1 PARA FUNL
7  CALL FUNX(T,H,F,UDT)
  GO TO 9
8  CALL FUNL(T,H,F,UDT)

```

```

9      DO 10 I=1,N11
      Z=A(J)*(AK(I)-B(J)*Q(I))
      Y(I)=Y(I)+H*Z
10     Q(I)=Q(I)+3*Z-C(J)*AK(I)
      RETURN
      END

C          CÁLCULO DO GRADIENTE
C
C      SUBROUTINE GRAD
C
C          MODO=1 := ESPAÇO DE FUNÇÕES CONTÍNUAS
C          MODO=2 := ESPAÇO DOS PARÂMETROS
C          MODO=3 := ESPAÇO DE ENTRADAS AMOSTRADAS
C          MODO=4 := ESPAÇO DE M.L.P.
C          MODO=5 := ESPAÇO DO USUÁRIO
C
      DIMENSION UL(201)
      COMMON U(201),G(201),X(10,201),ALBDA(10,201)
      COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
      COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
10     Q(N+1)=0
      DO 20 I=1,N
      Q(I)=0
20     X1(I)=X0(I)
      H=DT
      T=TI
      NP=NPTOSG+1
      NPT=NPTOS+1
      IF(MODO-4)11,12,11
12     DO 26 I=1,NPT
      CALL USUB(UL,T,U)
26     T=T+H
      T=TI
11     DO 30 I=1,NPT
      IF(MODO-4)27,28,27
28     CALL RUNGE(T,H,0.,UL)
      GO TO 29
27     CALL RUNGE(T,H,0.,U)
29     DO 25 J=1,N
25     X(J,I)=X1(J)
30     T=T+H
      CALL GRAXF
      N11=N+1
      DO 40 I=1,N11
40     Q(I)=0
      H=-DT
      T=T+H
      DO 70 I=1,NPT
      IF(MODO-4)31,32,31
32     CALL RUNGE(T,H,1.,UL)
      GO TO 33
31     CALL RUNGE(T,H,1.,U)
33     DO 60 J=1,N
      IAUXJ=NPTOS+2-I
      ALBDA(J,IAUXJ)=X1(J)
60     CONTINUE

```

```

70      T=T+H
      GO TO(80,100,130,160,210),MODO
C
C
C
80      DO 90 I=1,NP
      I1=I
      CALL GRADI(GA,I1,0)
90      G(I)=GA
      RETURN
C
C
C
      ESPAÇO DOS PARÂMETROS
C
C
100     DO 120 J=1,NP
      DO 110 I=1,NPT
      J1=J
      I2=I
      CALL GRADI(GA,I2,J1)
110     CT(I)=GA
      CALL SYMP(CT,ESC,NPTOS,DT)
120     G(J)=ESC
      RETURN
C
C
C
      ESPAÇO DE ENTRADAS AMOSTRADAS
C
C
130     NG=1
      NTE=NPTOS/M
      NPX=NPTOS-NTE
      DO 150 I=1,NPX,NTE
      I1=I
      DO 140 J=1,NTE
      CALL GRADI(GA,I1,NG)
      I1=I+J
140     CT(J)=GA
C
C
      DEVE INTEGRAR PARA UM NÚMERO ÍMPAR DE PONTOS
      J=J+1
      CALL GRADI(GA,I1,NG)
      CT(J)=GA
      CALL SYMP(CT,ESC,NTE,DT)
      G(NG)=ESC
150     NG=NG+1
      RETURN
C
C
C
      ESPAÇO DE MODULAÇÃO POR LARGURA DE PULSO (M.L.P.)
160     NTE=NPTOS/M
      DO 200 J=1,M
      IF(ABS(U(J))-1)212,211,211
211     AUX=1.
      GO TO 213
212     AUX=U(J)
213     TP=(J-1)*NTE+ABS(AUX)*NTE
      I2=TP+1
      IF(U(J))204,201,204
204     IF(ABS(U(J))+U(J))201,202,201
201     I3=1
      GO TO 203
202     I3=-1
C
C
      I2 FORNECE O ÍNDICE DE ALBDA

```



```

C          I3 É A FUNÇÃO SIGN
203 CALL GRADI(GA,I2,I3)
      G(J)=GA*NTE
200 CONTINUE
      RETURN

C
C          ESPAÇO DO USUÁRIO
C
210 CALL GRADI(GA,I2,I3)
      RETURN
      END

C
C          ESCSUB=PRODUTO ESCALAR
C          MODE=0 PROD. ESC. DE VETORES
C          MODE=1 PROD. ESC. DE FUNCOES
C
      SUBROUTINE ESCSU(CB1,CB,MODE,ESC,NDIM,DX)
      DIMENSION CB(201),CB1(201),CB2(201)
      COMMON U(201),G(201),X(10,201),ALBDA(10,201)
      COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
      ESC=0
      NDI=NDIM+1
      IF(MODE)10,10,20
10 DO 30 I=1,NDI
30 ESC=ESC+CB(I)*CB1(I)
      RETURN
20 DO 40 I=1,NDI
40 CB2(I)=CB(I)*CB1(I)
      CALL SYMP(CB2,ESC,NDIM,DX)
      RETURN
      END

C
C          SYMP = INTEGRAÇÃO PELA REGRA DE SYMPSON
C
C          O NÚMERO TOTAL DE PONTOS É IMPAR; N É PAR
C
      SUBROUTINE SYMP(Y,SY,N,DX)
      DIMENSION Y(201)
      C          DX=INTERVALO ENTRE CADA PONTO
      C          N=NÚMERO TOTAL DE PONTOS-1
      N11=N+1
      SY=Y(1)
      DO 10 I=3,N11,2
10 SY=SY+4*Y(I-1)+2*Y(I)
      SY=SY-Y(N+1)
      SY=SY*DX/3.
      RETURN
      END

C
C          BUSCA
C          NALF0 CONTA QUANTAS VEZES É CHAMADA A S/R
C          DURANTE AS ITERAÇÕES
C          SE NALF0=1, É A PRIMEIRA VEZ QUE SE FAZ A BUSCA
C
      SUBROUTINE BUSCA(ALFA,ALF0,NALF0)

```

```

        DIMENSION ALF0(201)
        COMMON U(201),G(201),X(10,201),ALBDA(10,201)
        COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
        COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
        NP=NPTOSG+1
        NPT=NPTOS+1
        CALL CUSTO(U,A1)
40      IF(NALF0-1) 60,50,60
50      CALL ESCSU(G,G,MODE,ESC,NPTOSG,DT)
        WRITE(*,555)ESC
555     FORMAT(' ESC= ',G16.8)
        B=.03*A1/ESC
        NSW=10
        WRITE(*,300)B
300     FORMAT(' BUSCA B= ',G16.8)
        GO TO 80
60      B=1
        ND=NALF0-1
        DO 70 I=1,ND
        WRITE(*,301)I,ALF0(I)
301     FORMAT(' BUSCA I= ',I4,' ALF01= ',G16.8)
        IF(ALF0(I)) 70,71,70
71      ALF0(I)=1.E-30
70      B=B*ALF0(I)**(1./ND)
        NSW=3
80      DO 90 I=1,NP
90      CT(I)=U(I)+B*SI1(I)
        CALL CUSTO(CT,A2)
        WRITE(*,302)A1,A2
302     FORMAT(' BUSCA A1= ',G16.8,' A2= ',G16.8)
        IF(A1-A2) 91,92,92
91      KM=-1
        GO TO 110
92      KM=1
110     K=0
        AD=1
        JCON=1
114     IF(AD) 117,117,115
115     AA=2.** (KM*K)*B
        AB=2.** (KM*(K+1))*B
117     AC=2.** (KM*(K+2))*B
        IF(AD) 135,135,118
118     DO 120 I=1,NP
120     CT(I)=U(I)+AA*SI1(I)
        CALL CUSTO(CT,AAT)
        DO 130 I=1,NP
130     CT(I)=U(I)+AB*SI1(I)
        CALL CUSTO(CT,ABT)
135     DO 140 I=1,NP
140     CT(I)=U(I)+AC*SI1(I)
        CALL CUSTO(CT,ACT)
        WRITE(*,303)AA,AB,AC,AAT,ABT,ACT
303     FORMAT(' BUSCA AA= ',G14.8,' AB= ',G14.8,' AC= ',G14.8,
1' AAT= ',G14.8,' ABT= ',G14.8,' ACT= ',G14.8)
        IF(AAT-ABT) 170,150,150
150     IF(ABT-ACT) 170,160,160

```

```

160 IF(A1-ACT) 161,295,161
295 JCON=JCON+1
    IF(JCON-NSW) 161,161,200
161 K=K+1
    AA=AB
    AB=AC
    AAT=ABT
    ABT=ACT
    AD=0
    GO TO 114
170 IF(AA-AB) 501,501,502
502 ACOP=AA
    ACOPT=AAT
    AA=AC
    AAT=ACT
    AC=ACOP
    AC1=ACOPT
501 DO 176 ICON=1,4
    DDK=.5*(AA+AB)
    DEK=.5*(AB+AC)
    DO 171 I=1,NP
171 CT(I)=U(I)+DDK*SI1(I)
    CALL CUSTO(CT,DDKT)
    DO 172 I=1,NP
172 CT(I)=U(I)+DEK*SI1(I)
    CALL CUSTO(CT,DEKT)
    WRITE(*,304) DDK,DEK,DDKT,DEKT
304 FORMAT(' BUSCA DDK=',G14.8,' DEK=',G14.8,' DDKT',G14.8,
1' DEKT=',G14.8)
    IF(AAT-DDKT) 175,174,174
174 IF(DDKT-ABT) 175,175,250
175 AC=AB
    ACT=ABT
    AB=DDK
    AB1=DDKT
    GO TO 176
250 IF(ABT-DEKT) 260,260,255
260 AA=DDK
    AAT=DDKT
    AC=DEK
    ACT=DEKT
    GO TO 176
255 AA=AB
    AAT=ABT
    AB=DEK
    ABT=DEKT
176 CONTINUE
    E1=AAT*(AC**2-AB**2)+ABT*(AA**2-AC**2)
    1+ACT*(AB**2-AA**2)
    E2=AAT*(AC-AB)+ABT*(AA-AC)+ACT*(AB-AA)
    WRITE(*,305) E1,E2
305 FORMAT(' BUSCA E1='G16.8,' E2=',G16.8)
    IF(E2) 181,182,181
182 E2=1.E-30
181 DD1=0.5*E1/E2
    DO 180 I=1,NP
180 CT(I)=U(I)+DD1*SI1(I)

```

```

CALL CUSTO(CT,DD1T)
WRITE(*,306)AB,DD1,ABT,DD1T
306  FORMAT(' BUSCA AB=',G16.8,' DD1=',G16.8,' ABT=',G16.8,
1 ' DD1T=',G16.8)
IF(ABT-DD1T)200,200,190
190  ALFA=DD1
RETURN
200  ALFA=AB
RETURN
END

C
SUBROUTINE CUSTO(BT,AC)
C      BT=VETOR DE ENTRADA DE NPTOSG COMPONENTES
DIMENSION BT(201),UL(201)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
NPT=NPTOS+1
Q(N+1)=0
DO 10 I=1,N
Q(I)=0
10  X1(I)=X0(I)
H=(TF-TI)/NPTOS
C      NPTOS=NÚMERO DE INTERVALOS ENTRE TI E TF,
C      E NAO DE PONTOS
T=TI
IF(MODO-4)50,12,50
12  DO 26 I=1,NPT
CALL USUB(UL,T,BT)
26  T=T+H
T=TI
50  DO 30 I=1,NPT
IF(MODO-4)29,28,29
28  CALL RUNGE(T,H,0.,UL)
GO TO 30
29  IF(IPGK-1)27,200,27
200 CALL PAG(BT,T,UL)
GO TO 28
27  CALL RUNGE(T,H,0.,BT)
30  T=T+H
C
C      X E Q SÃO AUTOMATICAMENTE INCREMENTADOS
C      PELA S/R RUNGE
C
CALL FI(AC)
C
RETURN
END
C
SUBROUTINE INITI
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
A(1)=.5

```

```

B(1)=2.
B(2)=1.
B(3)=1.
B(4)=2.
C(1)=.5
C(4)=.5
N11=N+1
A(2)=1.-SQRT(.5)
A(3)=1.+SQRT(.5)
A(4)=1./6.
C(2)=A(2)
C(3)=A(3)
DT=(TF-TI)/NPTOS
IF(MODO-4) 10,20,10
20 ALP=1.
10 CONTINUE
RETURN
END
SUBROUTINE USUB(ULA,T,UDT)
DIMENSION UDT(201),ULA(201)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
NTE=NPTOS/M
I=(T-TI)/DT
K=I/NTE+1
AUX=UDT(K)
I1=I+1
IF(UDT(K)) 61,21,61
61 IF(ABS(UDT(K))-1) 50,60,60
60 AUX=1
UDT(K)=UDT(K)/ABS(UDT(K))
50 IF((ABS(AUX)*NTE)+(K-1)*NTE-I) 21,21,10
10 IF(ABS(UDT(K))+UDT(K)) 30,40,30
30 ULA(I1)=1.
GO TO 20
40 ULA(I1)=-1
GO TO 20
21 ULA(I1)=0.
20 CONTINUE
RETURN
END
SUBROUTINE PAGI(P1,P2,NP)
DIMENSION P1(201),P2(201)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
IF(IPGK-1) 80,20,80
20 DO 50 I=1,NP
IF(ABS(U(I))-ALP) 30,40,40
30 P1(I)=P2(I)
GO TO 50
40 P1(I)=0.
50 CONTINUE

```

```
      GO TO 70
80    DO 60 I=1,NP
60    P1(I)=P2(I)
70    CONTINUE
      RETURN
      END
SUBROUTINE PAG(BT,T,UL)
      DIMENSION BT(201),UL(201)
      RETURN
      END
```

APÊNDICE B2

LISTAGEM DE UM DOS SUBPROGRAMAS DE MINIMIZAÇÃO PELO MÉTODO DO GRADIENTE CONJUGADO - PARTE DEPENDENTE DA APLICAÇÃO -

O algoritmo a seguir constitui a parte dependente do sistema de solução para o problema de controle ótimo em tempo real com tempo final livre sobre o espaço de funções contínuas.

Ele especifica as condições iniciais do sistema e controla a seqüência de execução do método do gradiente conjugado. Para isso, o algoritmo passa o controle ao programa "Gramin" do *Apêndice B1*, o qual faz uso de subrotinas dependentes do problema, declaradas no programa da página seguinte.

C *Supõe U(t) inicial igual a 0 (UI = 0) e DT = 5seg.*
C

```
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11
```

C
C *COMMONS DO USUÁRIO*
C

```
COMMON R,W(3)
```

C
C *INICIALIZAÇÃO*
C

```
MODE=1
MODO=1
N=4
M=1
WRITE(*,333)
333  FORMAT(' ENTRE COM WI,R')
    READ(*,334)WI
    READ(*,334)R
334  FORMAT(G16.8)
    WRITE(*,335)WI
335  FORMAT(' WI= ',G16.8)
    W(1)=WI
    W(2)=WI
    W(3)=WI
    TI=0.
    TF=5.
    DT2=TF-TI
    WRITE(*,336)
336  FORMAT(' ENTRE COM NITER ')
    READ(*,337)NITER
337  FORMAT(I3)
    WRITE(*,338)NITER
338  FORMAT(' NITER= ',I3)
    WRITE(*,339)
339  FORMAT(' ENTRE COM NPTOS, NIV ')
    READ(*,337)NPTOS
    READ(*,337)NIV
    WRITE(*,340)NPTOS
340  FORMAT(' NPTOS= ',I3)
    NPTOSG=NPTOS
    DO 10 I=1,N
10   X0(I)=0.
    WRITE(*,341)
341  FORMAT(' ENTRE COM U0 ')
    READ(*,334)UI
    WRITE(*,342)UI
342  FORMAT(' U0= ',G16.8)
    DO 20 I=1,201
20   U(I)=UI
    NPT=NPTOS+1
    WRITE(*,345)
345  FORMAT(' ENTRE COM IRREST ')
    READ(*,337)IRT
```



```

346 WRITE(*,346) IRT
    FORMAT(' IRREST= ',I3)
C
C      FIM DA INICIALIZAÇÃO
C
    NIVCONT=0
    OPEN(66,FILE='FCV3.DAT',STATUS='UNKNOWN')
    DO 42 IC=1,NIV
    DO 41 IRREST=1,IRT
    CALL GRAMI
41  CONTINUE
    DO 30 I=1,NPT
    I2=NIVCONT+I
30  WRITE(66,100) I2,X(1,I),X(2,I),X(3,I),X(4,I),U(I)
100  FORMAT(2X,I4,2X,5G20.8)
    WRITE(*,354) NITER,NPTOS,W(1),IRT
354  FORMAT(2X,' NITER= ',I5,' NPTOS= ',I5,' WI= ',G14.8,
1' IREST1= ',I5)
    NIVCONT=NIVCONT+NPTOS
    TI=TF
    TF=TF+DT2
    DO 43 I=1,N
43  X0(I)=X(I,NPT)
    DO 44 I=1,201
    U(I)=UI
44  CONTINUE
42  CONTINUE
    END
C
C      PARTE DEPENDENTE DAS EQUAÇÕES DO SISTEMA
C
C
C      funx = descrição das equações do sistema
C      (eq.3.3.2-10 à 3.3.2-13)
C
    SUBROUTINE FUNX(T,H,F,UDT)
    DIMENSION F(11),UDT(201)
    COMMON U(201),G(201),X(10,201),ALBDA(10,201)
    COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
    COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
    COMMON A(4),B(4),C(4),N11
C
C      COMMONS DO USUÁRIO
C
    COMMON R, W(3)
    I=(T-TI)/DT+1
    F(1)=X1(2)
    F(2)=-X1(2)/3.+5.*X1(3)
    F(3)=-X1(3)+.1*UDT(I)
    F(4)=(X1(1)-10)**2+R*UDT(I)**2
    RETURN
    END
C
C      Fun1 = descrição das equações adjuntas
C      (eq. 3.3.2-15 à 3.3.2-18)
C

```

```

SUBROUTINE FUNL(T,H,F,UDT)
  DIMENSION F(11),UDT(201)
  COMMON U(201),G(201),X(10,201),ALBDA(10,201)
  COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
  COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
  COMMON A(4),B(4),C(4),N11

```

COMMONS DO USUÁRIO

```

COMMON R,W(3)

```

```

I=(T-TI)/DT+1
F(1)=-2*(X(1,I)-10)*X1(4)
F(2)=-1*(X1(1)-1./3.*X1(2))
F(3)=-1*(5*X1(2)-X1(3))
F(4)=0
RETURN
END

```

*Condição terminal para a equação adjunta
(eq. 3.3.2-19)*

```

SUBROUTINE GRAXF
  COMMON U(201),G(201),X(10,201),ALBDA(10,201)
  COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
  COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
  COMMON A(4),B(4),C(4),N11

```

COMMONS DO USUÁRIO

```

COMMON R,W(3)

X1(1)=2.*W(1)*(X1(1)-10.)
X1(2)=2.*W(2)*X1(2)
X1(3)=2.*W(3)*X1(3)
X1(4)=1.
RETURN
END

```

*Expressão do gradiente em função da equação adjunta
(eq. 3.3.2-14)*

```

SUBROUTINE GRADI(GA,I,J)
  COMMON U(201),G(201),X(10,201),ALBDA(10,201)
  COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
  COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
  COMMON A(4),B(4),C(4),N11

```

COMMONS DO USUÁRIO

```

COMMON R,W(3)

GA=.1*ALBDA(3,I)+2*R*U(I)*ALBDA(4,I)
RETURN

```

```

END
SUBROUTINE FI(AC)
COMMON U(201),G(201),X(10,201),ALBDA(10,201)
COMMON N,M,NPTOS,NITER,CT(201),CT1(201),SI1(201),IPGK,
1ALP,ID
COMMON X0(10),TI,TF,DT,NPTOSG,MODE,MODO,MX,X1(11),Q(11)
COMMON A(4),B(4),C(4),N11

```

COMMONS DO USUÁRIO

```

COMMON R,W(3)

```

```

AC=X1(4)+W(1)*(X1(1)-10.)**2+W(2)*X1(2)**2+W(3)*X1(3)**2
RETURN
END

```