UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Controle ótimo de descarregadores de navio

Marco Antonio Oliveira da Silva

Monografia orientada pelo Prof. José Jaime da Cruz (POLI-USP) e apresentada à Universidade de São Paulo, como parte dos requisitos necessários para a conclusão do curso de Bacharelado em Matemática Aplicada.

São Paulo - SP
2007

Agradecimentos

Quero agradecer primeiramente à Deus, por ter me abençoado e me dado forças para lutar até este momento, pois é nele em quem eu penso quando as coisas não estão bem e quando sou grato por algo.

Quero agradecer à minha família, que sempre teve paciência comigo e tem estado por perto e me dado atenção quando eu realmente precisei. Infelizmente meu pai não pôde presenciar este grande momento da minha vida, pois ele não está mais entre nós, mas sei que ficaria orgulhoso de mim.

Quero agradecer ao meu orientador, o Professor José Jaime da Cruz (PTC-POLI), um excelente profissional e uma excelente pessoa, por ter me orientado a realizar meu trabalho de conclusão de curso e quem eu tenho uma grande admiração.

Quero agradecer em especial à Professora Sônia Regina Leite Garcia (MAP-IME), uma excelente profissional, uma excelente pessoa e uma grande amiga que me ajudou e participou várias vezes da minha vida acadêmica e pessoal.

Quero agradecer aos meus amigos que estiveram ao meu lado nos momentos de tristezas e alegrias cujas amizades eu fiz durante minha permanência na universidade e provavelmente continuarei amigo de alguns destes após minha saída.

Quero também agradecer em especial ao meu amigo Felipe Pavanello Sultani, por ter sido meu maior companheiro acadêmico e profissional. Uma pessoa que tenho uma grande admiração.

Finalmente, quero agradecer à Universidade de São Paulo, uma Universidade de renome, por sua excelência e pelo suporte às minhas necessidades.

Sumário

1 Introdução

Guindastes são usados em plantas industriais, armazéns, portos e locais em construção onde cargas pesadas devem ser transferidas por longas distâncias. Existem vários tipos de guindastes e várias formas de utilizá-los. Em portos, onde o descarregamento depende de guindastes, a velocidade da operação é o principal objetivo na projeção de guindastes. Em muitos casos somente a habilidade do operador humano limita a velocidade máxima, porém, em transferências cuidadosas e posicionamento exato da carga são preocupantes.

Os guindastes suspensos têm sido um objeto de estudos de controle, especialmente ao carregamento e ao descarregamento de galpão (container) de navios. Uma revisão da literatura mostra que o problema ainda é uma ativa área de pesquisa, embora existem poucos artigos sobre os problemas do sistema de controle de guindastes. Auernig e Troger [1] usaram controle de tempo mínimo para minimizar o balanço da carga. Corriga et al. (apud [2]) aplicaram um método de programação de ganho implícito para controlar o guindaste. Alguns pesquisadores também usaram o modelo dinâmico do guindaste para avaliar uma referência de velocidade ótima que minimiza o balanço da carga (apud [2]). Entretanto, desde que o balanço da carga depende do movimento e da aceleração do carro, minimizar o tempo de ciclo e minimizar o balanço da carga são parcialmente exigências conflitantes. Singhose et al. (apud [2]) propôs o método de entrada que dá forma ao controle do guindaste. Alguns pesquisadores também aplicaram a teoria de controle não-linear para analisar as propriedades do sistema de guindaste (apud [2]). Estes métodos são demasiado complexos executar para o uso da indústria. Moustafa e Ebied (apud [2]) usaram um controle anti-balanço para os guindastes suspensos. O foco destes pesquisadores é o controle na supressão do balanço da carga mas não resolvem o problema do erro da posição no movimento do guindaste. Alguns métodos baseados em fuzzy (apud [2]) também foram propostos

para o controlar o guindaste. Infelizmente, tais controladores fuzzy não podem fornecer o desempenho desejado ao sistema do guindaste, devido à incerteza e aos grandes distúrbios do sistema fuzzy, reduzindo a eficiência do trabalho.

O trabalho de um descarregamento de cargas deve ter o mínimo tempo possível, e para isso precisa de altas velocidades e consequentemente de altas acelerações. O balanço da carga suspensa é indesejável nos pontos extremos do percurso, pois na saída do container poderia danificar o equipamento com eventuais colisões e na chegada à moega também poderia despediçar material. Conseqüentemente, é uma exigência fundamental procurar um método de controle satisfatório que faça com que o balanço no fim da transferência da carga no tempo final seja nulo. Também, minimizar o tempo de tal transferência trará um custo menor às operações de tais transferências.

Economicamente, este estudo é de grande importância, pois além de terminar o processo de descarregamento em menos tempo, o local fica disponibilizado para efetuar novas operações.

A idéia central deste trabalho, é de restringir as condições do movimento do carro para que a carga saia do container e chegue na moega sem oscilação na saída e na chegada no menor tempo possível, dados a função de deslocamento vertical (ver Apêndice A) da carga que é puxada pelo cabo e os parâmetros envolvidos no processo.

A caçamba (figura 1) tem um ponto fixo de posição com respeito ao navio e assim o processo de descarregamento é feito com o deslocamento horizontal do carrinho que levanta a carga e leva até a moega onde é descarregada. Matematicamente este processo otimizado deve ser resolvido.

Aqui será feito um modelo físico que representa o sistema do descarregamento, pelas equações de Lagrange, de onde estão relacionadas as energias cinéticas e potenciais. Para efeitos de simulações computacionais, estes modelos serão discreti-

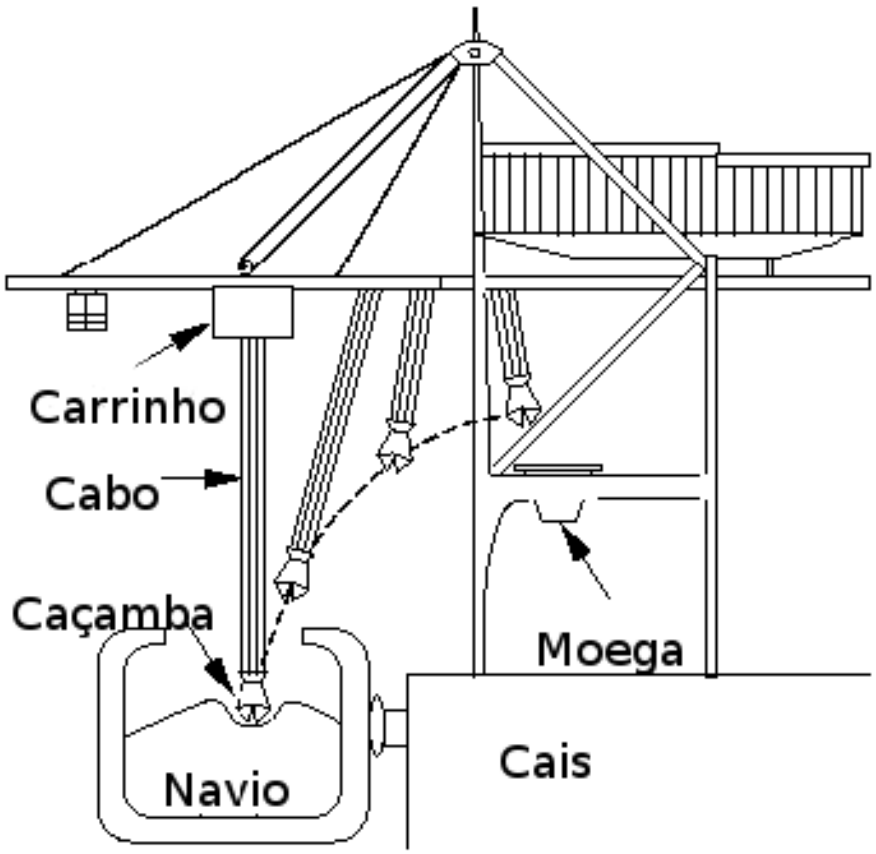

Figura 1: Sistema de descarregamento

zados na forma de estados $x(k+1) = Fx(k)+Gu(k)$ e otimizados via Programação Linear.

No modelo serão desconsiderados a deformação elástica da caçamba, os efeitos dissipativos como a resistência da rolagem, perdas no mecanismo de direção e as forças do vento, assumindo também durabilidade infinita de todos os elementos.

2 Modelo Dinâmico do Sistema

2.1 Formulação do Sistema

A figura (2) mostra o sistema mecânico do movimento do carrinho, do cilindro e da carga do descarregador. Este sistema possui três graus de liberdade x_1, ϕ e l.

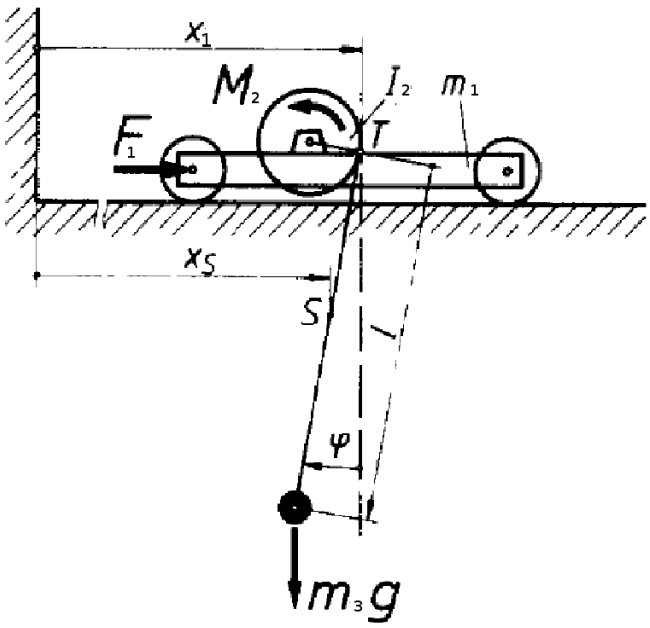

Figura 2: Sistema mecânico, com 3 graus de liberdade

Para a identificação das variáveis de seus respectivos corpos, são usados os seguintes índices:

1, carrinho

2, cilindro que puxa e solta o cabo

3, carga

Usando o carrinho como referência na origem do plano cartesiano (x,y) e o ângulo do cabo com origem entre o terceiro e o quarto quadrante em sentido

horário, obtem-se:

$$\begin{aligned} x_c &= x_1 - l\sin\phi \\ y_c &= -l\cos\phi \end{aligned} \tag{1}$$

onde o índice c do ponto (x_c, y_c) representa a posição da carga.

Equações da Energia Cinética T para cada corpo:

$$\begin{aligned} T_1 &= \frac{m_1}{2}(\dot{x}_1^2) \\ T_2 &= \frac{I\omega^2}{2} = \frac{m_2}{2}(\dot{l}^2) \\ T_3 &= \frac{m_1}{2}(\dot{x}_c^2 + \dot{y}_c^2) \end{aligned} \tag{2}$$

Equações da Energia Potencial π para cada corpo:

$$\begin{aligned} \pi_1 &= 0 \\ \pi_2 &= 0 \\ \pi_3 &= m_3 g y_c \end{aligned} \tag{3}$$

Substituindo as variáveis em seus respectivos lugares (2 e 3), podemos obter a Lagrangeana L com os T's e os π's:

$$\begin{aligned} L &= L(x_1, l, \phi, \dot{x}_1, \dot{l}, \dot{\phi}) \\ L &= \sum_{i=1}^{3} T_i - \sum_{i=1}^{3} \pi_i \\ L &= \frac{m_1}{2}\dot{x}_1^2 + \frac{m_2}{2}(\dot{l}^2) + \frac{m_3}{2}(\dot{x}_1^2 + \dot{l}^2 + l^2\dot{\phi}^2 \\ &\quad -2\dot{x}_1\dot{l}\sin\phi - 2\dot{x}_1 l\cos\phi\dot{\phi}) + m_3 g l\cos\phi \end{aligned}$$

Sejam q_i com $i = 1, 2, 3$ as coordenadas generalizadas, onde $q_1 = x_1$, $q_2 = l$, $q_3 = \phi$ e Q_i as forças F_i aplicadas no corpo i , então podemos descrever as Equações de Euler-Lagrange na seguinte forma:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \tag{4}$$

logo, pela substituição das variáveis citadas nas equações (4) e linearizando o sistema com $\sin\phi = \phi$ e $\cos\phi = 1$ (ϕ e $\dot{\phi}$ são pequenos o suficiente para serem linearizados na origem, devido ao estado estacionário e a amplitude do balanço), obtemos

$$\ddot{x}_1 + \frac{m_3}{m_1}(\ddot{x}_1 - \ddot{l}\phi - 2\dot{l}\dot{\phi} - l\ddot{\phi}) = \frac{F_1}{m_1} \tag{5}$$

$$-\left(1 + \frac{m_3}{m_2}\right)\ddot{l} + \frac{m_3}{m_2}\left(\ddot{x}_1\phi + g\right) = -\frac{F_2}{m_2} \tag{6}$$

$$2\dot{l}\dot{\phi} + l\ddot{\phi} + g\phi = \ddot{x}_1 \tag{7}$$

As seguintes variáveis de entrada são atribuídas para o sistema controlado por posição e o sistema controlado por força respectivamente

$$u_1 = \frac{a_1(t)}{a_{1_{max}}} \text{ ou } u_1 = \frac{F_1(t)}{F_{1_{max}}} \tag{8}$$

onde para o sistema controlado da posição

$$a_{1_{max}} = max(|a_1(t)|) \tag{9}$$

e para o sistema controlado da força

$$a_{1_{max}} = \frac{F_{1_{max}}}{m_1} \ , \ F_{1_{max}} = max(|F_1(t)|) \tag{10}$$

Atribuiremos às seguintes variáveis as quantidades adimensionais

$$\begin{aligned} \tau &= \omega_{max} t \ , \ \omega_{max} = \sqrt{\frac{g}{l_{min}}} \\ \sigma &= \frac{x_1 g}{a_{1_{max}} l_{min}} \\ \Phi &= \frac{g}{a_{1_{max}}} \phi \\ \lambda &= \frac{l}{l_{min}} \end{aligned} \tag{11}$$

onde g é a força da gravidade, $l_{min} = min(|l(t)|)$ e $l_{max} = max(|l(t)|)$.

Com $()' = \frac{d}{d\tau}$, as equações de movimento (5,6 e 7) com as variáveis adimensionalizadas são apresentadas como

$$\frac{F_1}{m_1 a_{1_{max}}} = \sigma'' + \frac{m_3}{m_1}(\sigma'' - \Phi\lambda'' - 2\lambda'\Phi' - \lambda\Phi'') \tag{12}$$

$$-\frac{F_2}{m_2} = -\big(1 + \frac{m_3}{m_2}\big) g\lambda'' + \frac{m_3}{m_2}\big(\frac{a^2_{1_{max}}}{g}\sigma''\Phi + g\big) \tag{13}$$

$$\sigma'' = 2\lambda'\Phi' + \lambda\Phi'' + \Phi \tag{14}$$

Inserindo 14 em 12 e substituindo valores em 14, temos:

$$u_1 = 2\lambda'\Phi' + \lambda\Phi'' + \Phi(1 + \alpha(1 - \lambda''))$$

$$-\frac{F_2}{m_2 g} = -\big(1+\frac{m_3}{m_2}\big)\lambda'' + \frac{m_3}{m_2}\big(\frac{a^2_{1_{max}}}{g^2}\sigma''\Phi + 1\big)$$

$$u_1 = \sigma'' + \alpha\Phi(1-\lambda'') \tag{15}$$

Essas equações são válidas tanto para controle por posição quanto pela força do sistema controlado.

$\alpha = \mu = \frac{m_3}{m_1}$ pode ser usado como parâmetro para distinguir estes dois casos em (15). Quando se trata de controle por posição, o sistema controlado tem dois graus de liberdade (Φ e λ) porque σ é prescrita. Daqui não há acoplamento entre σ e Φ. Para o sistema controlado por posição, $\alpha = 0$ deve ser atribuída em (15).

Na prática, $\lambda'' < 0,2 << 1$ é válida (apud [1]) em operações de descarregamento e o tempo de aceleração é muito pequeno em relação ao tempo do movimento do carrinho. A influência da aceleração do cabo que puxa a carga no balanço da carga é insignificante e a suposição de uma velocidade constante no levantamento da carga fica bem justificada.

Seja v_2 uma constante (adimensionalizada) da velocidade de içamento da carga

$$v_2 = \frac{V_2}{\sqrt{g l_{min}}} \tag{16}$$

como adotaremos o sistema controlado da posição ($\alpha = 0$), as equações de movimento (15) ficam

$$\begin{aligned} u_1 &= 2\lambda'\Phi' + \lambda\Phi'' + \Phi \\ u_1 &= \sigma'' \\ v_2 &= \lambda' \end{aligned} \tag{17}$$

2.2 Discretização

A primeira equação das equações (17) deve ser discretizada para ser tratada computacionalmente.

Seja $\lambda(\tau)$ uma função dada arbitrariamente (ver Apêndice A). Então Φ deve ser a solução de 17 e $(\Phi(\tau), \Phi'(\tau))$ nos dá o plano de fases do sistema.

Seja $\mathbf{z} \in \mathbf{R^2}$, tal que $z_1 = \Phi$ e $z_2 = \Phi'$ então

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= \frac{u_1}{\lambda} - \frac{2\lambda' z_2}{\lambda} - \frac{z_1}{\lambda} \end{aligned} \tag{18}$$

que podemos escrever da forma

$$\mathbf{z}' = \begin{pmatrix} 0 & 1 \\ -\frac{1}{\lambda} & -\frac{2\lambda'}{\lambda} \end{pmatrix} \mathbf{z} + u_1 \begin{pmatrix} 0 \\ \frac{1}{\lambda} \end{pmatrix} \tag{19}$$

portanto temos um sistema de equações diferenciais de primeira ordem do tipo

$$\mathbf{z}' = \mathbf{A}(\tau)\mathbf{z} + \mathbf{B}(\tau)u \tag{20}$$

Pela proposição 3.5 (Doering, C. J. apud [5]), seja $\mathbf{X} : \mathbf{I} \to \mathbf{M}(n)$ uma matriz fundamental de $\mathbf{z}' = \mathbf{A}(t)\mathbf{z}$ e sejam $t_0 \in \mathbf{I}$ e $\mathbf{z_0} \in \mathbf{R^n}$ dados. Então

$$\mathbf{z}(t) = \mathbf{X}(t)[\mathbf{X}(t_0)]^{-1}\mathbf{z_0} + \mathbf{X}(t)\int_{t_0}^{t} [\mathbf{X}(s)]^{-1}\mathbf{b}(s)ds \tag{21}$$

é a única solução de $\mathbf{z}' = \mathbf{A}(t)\mathbf{z} + \mathbf{b}(t)$ tal que $\mathbf{z}(t_0) = \mathbf{z_0}$.

Não existem métodos gerais para a obtenção de matrizes fundamentais $\mathbf{X}(t)$ das equações homogêneas $\mathbf{z}' = \mathbf{A}(t)\mathbf{z}$, necessárias para a utilização da fórmula e

não existe uma forma fechada geral para escrever as soluções de sistemas lineares não-autônomos. Por isso devemos resolver numericamente o sistema e para isso devemos discretizá-lo.

Sabemos que (20) são as equações de estado do sistema, onde

$$\mathbf{z}(\tau) = \mathbf{\Phi}(\tau, \tau_0)\mathbf{z}(\tau_0) + \mathbf{\Phi}(\tau, \tau_0) \int_{\tau_0}^{\tau} [\mathbf{\Phi}(\gamma, \tau_0)]^{-1}\mathbf{B}(\gamma)u(\gamma)d\gamma \tag{22}$$

é a solução de 20 sendo $\mathbf{\Phi}$ a solução de

$$\begin{cases} \frac{\partial \mathbf{\Phi}(\tau,\tau_0)}{\partial \tau} = \mathbf{A}(\tau)\mathbf{\Phi}(\tau, \tau_0) \\ \mathbf{\Phi}(\tau, \tau_0) = \mathbf{I} \end{cases} \tag{23}$$

Discretizando 22 temos

$$\mathbf{z}(\tau_{k+1}) = \mathbf{\Phi}(\tau_{k+1}, \tau_k)\mathbf{z}(\tau_k) + \mathbf{\Phi}(\tau_{k+1}, \tau_k) \int_{\tau_k}^{\tau_{k+1}} [\mathbf{\Phi}(\gamma, \tau_k)]^{-1}\mathbf{B}(\gamma)u(\gamma)d\gamma \tag{24}$$

3 Formulação do Problema de Otimização

Devemos minimizar o tempo t da transferência da carga, entretanto para conseguirmos obtê-lo, devemos achar o maior percurso possível entre a distância do navio e a moega. A velocidade média é maior quando o percurso também é maior entre um ponto e outro fixos num mesmo intervalo de tempo.

Dado $S_{1_{final}}$ (ver significado dos parâmetros na seção "Simulações e Resultados"), precisamos minimizar t_{final}. Como $S_{1_{final}} = v_{1_{mdia}} t_{final}$, podemos fixar t_{final} e achar o máximo valor de $v_{1_{mdia}}$ tal que $S_{1_{final}}$ seja o mais próximo possível do percurso desejado do carro (distância entre o navio e a moega). Para tal procedimento, é realizada uma sequência de PPL's (ver Apêndice B) $(P_1, P_2, ...)$ fixando um tempo t a cada problema até que o percurso maximizado seja tão próximo do desejado quanto se queira. Este t é o tempo mínimo de tal transferência.

As variáveis de controle $u_1(i)$ são as variáveis que maximizam o percurso $S_{1_{final}}$, portanto **u** ótimo é o vetor utilizado nas simulações.

3.1 Função Objetivo

Nossa função objetivo é a funcão posição do carrinho representado como a variável admensional "Sigma"(σ). Como $\sigma'' = u_1$, a função objetivo é a integral de u_1 duas vezes, sendo que o resultado da primeira integração é a velocidade do carro.

$$\sigma(n) = \frac{(\Delta\tau)^2}{2} \sum_{i=1}^{n} [2(n-i)+1] u_1(i) \tag{25}$$

3.2 Restrições

Algumas limitações do sistema são impostas para que a solução esteja dentro de suas especificações. Estas restrições são utilizadas quando há um problema de

programação linear (PPL).

A seguir, serão apresentadas as restrições do nosso PPL:

- Aceleração ou função entrada

  Como $|u_1| \leqslant 1$ (8), todas as variáveis $u_1(i), i = 1, ..., 100$ estão entre -1 e 1 ($-1 \leqslant u_1(i) \leqslant 1$).

- Posição e velocidade angular

  Seja $\mathbf{x}(\tau) = \binom{\Phi(\tau)}{\Phi'(\tau)}$, então podemos representar $\mathbf{x}(\tau)$ como

$$\mathbf{x}(\tau_k) = \phi(\tau_k, 0)\mathbf{x}(0) + \int_0^{\tau_k} \phi(\tau_k, \xi)\mathbf{b}(\xi)u_1(k)(\xi)d\xi \tag{26}$$

  que pode ser parcelada em uma somatória de integrais de intervalos de integração $\Delta\tau$

$$\begin{aligned}\mathbf{x}(\tau_k) &= \phi(\tau_k, 0)\mathbf{x}(0) + \int_0^{\tau_1=\Delta\tau} \phi(\tau_k, \xi)\mathbf{b}(\xi)u_1(1)(\xi)d\xi \\ &+ \int_{\tau_1=\Delta\tau}^{\tau_2=2\Delta\tau} \phi(\tau_k, \xi)\mathbf{b}(\xi)u_1(2)(\xi)d\xi \\ &+ ... + \int_{\tau_{k-1}=(k-1)\Delta\tau}^{\tau_k=k\Delta\tau} \phi(\tau_k, \xi)\mathbf{b}(\xi)u_1(k)(\xi)d\xi \end{aligned} \tag{27}$$

  Sabemos que $\mathbf{x}(0) = 0$ que é a condição inicial do sistema, então $\phi(\tau_k, 0)\mathbf{x}(0) = \mathbf{0}$. Agora consideremos também $\Gamma_k(i) = \int_{\tau_{i-1}}^{\tau_i} \phi(\tau_k, \xi)\mathbf{b}(\xi)d\xi$, logo

$$\mathbf{x}(\tau_k) = \sum_{i=1}^{k} \Gamma_k(i)u_1(i) \tag{28}$$

  portanto

$$\mathbf{x}(\tau_f) = 0 \Leftrightarrow \sum_{i=1}^{n} \Gamma_n(i) u_1(i) = 0 \tag{29}$$

- Velocidade máxima do carro

  Sabemos que $\sigma''(\tau) = u_1$ e integrando-a obtemos a velocidade adimensional $\sigma'(\tau) = v_1(k)$

$$v_1(k) = \Delta\tau \sum_{i=1}^{k} u_1(i) \tag{30}$$

  onde k=1, 2, ..., n e $v_{1_{max}}$ é dada arbitrariamente.

  Se $|v_1(k)| \leqslant v_{1_{max}}$, então

$$\begin{aligned} -v_{1_{max}} \leqslant v_1(k) \leqslant v_{1_{max}} \Rightarrow \\ \Rightarrow \frac{-v_{1_{max}}}{\Delta\tau} \leqslant \sum_{i=1}^{k} u_1(i) \leqslant \frac{v_{1_{max}}}{\Delta\tau} \end{aligned} \tag{31}$$

  onde k=1, 2, ..., n-1 e quando k=n, temos que

$$\sum_{i=1}^{n} u_1(i)\Delta\tau = 0 \tag{32}$$

  que impõe estado de repouso no instante final.

A partir destas restrições e da função objetivo, podemos resolver o PPL na próxima seção.

3.3 Problema de Tempo Mínimo

Podemos obter $u_1(i)$ quando maximizamos o percurso do carro σ num período de tempo ótimo. Dadas as restrições e a função objetivo, temos

$$
\begin{aligned}
\max \quad & \sigma(n) = \frac{(\Delta t)^2}{2} \sum_{i=1}^{n} [2(n-i)+1] u_1(i) \\
\text{suj. a} \quad & -1 \leqslant u_1(i) \leqslant 1 \text{ (i = 1,...,n)} \\
& \sum_{i=1}^{n} \Gamma_n(i) u_1(i) = 0 \\
& \frac{-v_{Tmax}}{\Delta\tau} \leqslant \sum_{i=1}^{k} u_1(i) \leqslant \frac{v_{Tmax}}{\Delta\tau} \text{ (k = 1, ..., n-1)} \\
& \sum_{i=1}^{n} u_1(i) \Delta\tau = 0 \qquad (33)
\end{aligned}
$$

o PPL que pode obter solução ótima para o problema em questão.

4 Simulações e Resultados

Para realizar as simulações (ver código fonte em Apêndice F e Apêndice G) foram utilizados a linguagem C e o software livre GNUplot para mostrar os gráficos gerados pelo programa.

Os cálculos da simulação numérica foram feitos com a *Regra de Simpson* (ver Apêndice C) para calcular as integrações numéricas e o Método de Runge-Kutta de ordem dois conhecido como *Método do Ponto Médio* (ver Apêndice D) para dar a solução de ϕ_1 e ϕ_2.

Os parâmetros utilizados no problema são de protótipo de labaratório, por terem dimensões menores. Existem estudos experimentais de tais dimensões que possam confirmar a validação deste estudo teórico. Segue a tabela de parâmetros utilizada em nossos cálculos e simulações:

Variável	Valor	Significado
l_f	0,125 m	comprimento final (moega)
l_i	0,25 m	comprimento inicial (navio)
$a_{1_{max}}$	2,5 m/s^2	aceleração máxima do carro
n	100	número de sub-intervalos (iterações) entre a e b
$S_{1_{final}}$	0,25 m	percurso desejado do carro
t_{final}	2,2 s	instante final
$v_{2_{max}}$	0,5 m/s	velocidade de içamento da caçamba
$v_{1_{max}}$	0,12 m/s	velocidade máxima do carro

Além da regra de Simpson e do método do Ponto Médio para a resolução do sistema de equações diferenciais, uma biblioteca do software livre GLPK (GNU Linear Programming Kit) foi utilizado para a resolução do PPL. Este software é um pacote de rotinas escrito em linguagem C para resolução em larga escala de

problemas relacionados à programação linear. Também foi utilizado o software Matlab somente para a otimização em questão.

4.1 Matlab X GLPK

O Matlab, mais conhecido por sua robustez e por sua eficiência há muitos anos no mercado de infomática, com pacotes para cálculos específicos que podem ser incorporados às rotinas pré-definidas, foi utilizado para resolver o PPL assim como a biblioteca do GLPK, desenvolvido por programadores pelo projeto GNU com boa eficiência em otimização, porém no nosso problema, tem arredondado valores muito pequenos.

Pela figura (3), podemos observar que as duas funções não são iguais, devido ao arredondamento do glpk. Os valores de u_i $(i = 1, 2, ..., 100)$ podem ser vistos no Apêndice E.

O tempo entre a saída da origem e a chegada ao destino do carro e da carga, deve ser o tempo mínimo sem oscilação indesejada no final do trajeto (quando parados). O Matlab tem se mostrado mais preciso em seus resultados, obtendo números muito pequenos a cada variável ótima.

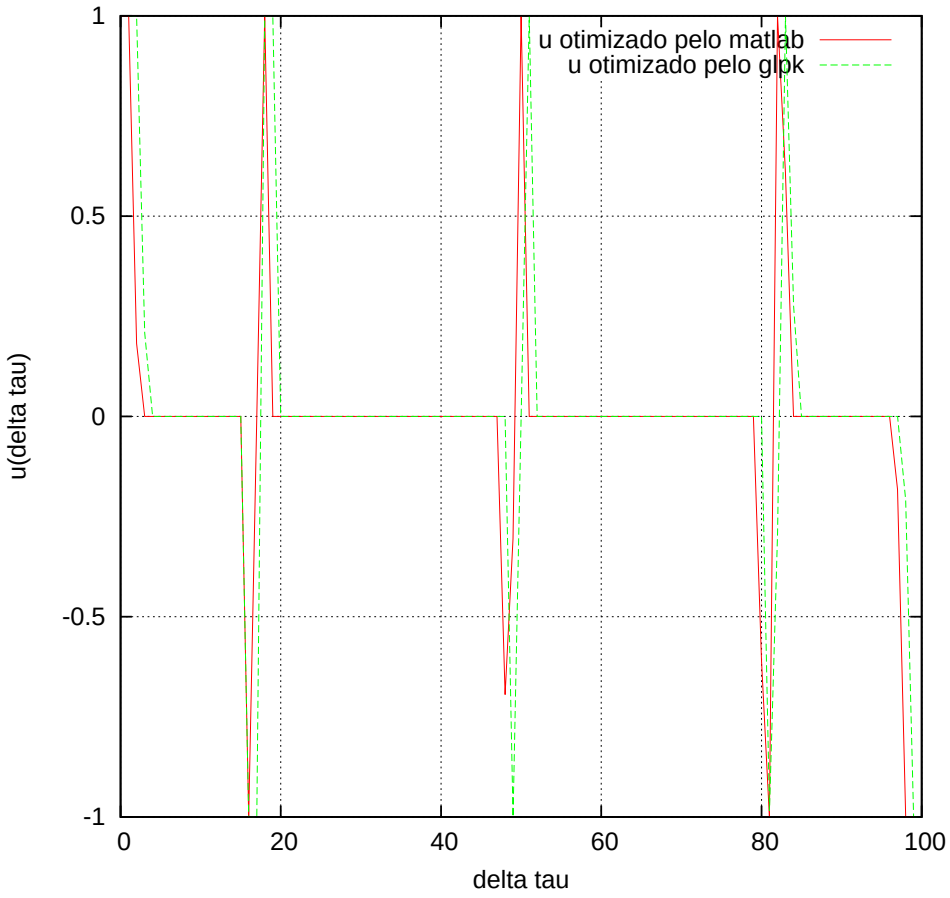

Figura 3: $u(\tau)$ otimizado pelo matlab e $u(\tau)$ otimizado pelo glpk

4.2 Otimização pelo Matlab

A seguir, mostraremos os gráficos de cada função relacionada aos três graus de liberdade $(\phi(t), \phi'(t), x_1(t), x_1'(t), x_1''(t))$ do nosso sistema mecânico.

4.2.1 Posição Angular da Carga

A posição angular da carga está representada pela função $\phi(t)$, medida em graus (°).

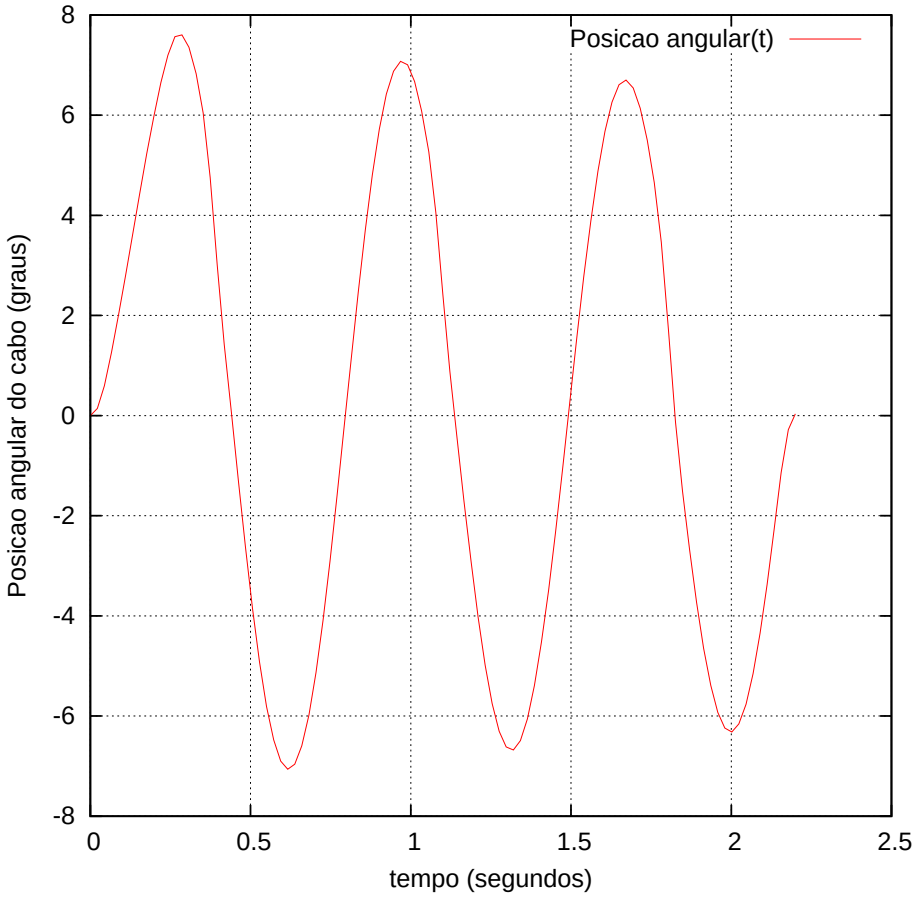

Figura 4: Posição angular da carga $t \times \phi(t)$

Pelo gráfico (4), podemos notar que a carga sai de seu estado em repouso, oscila entre $-8°$ e $8°$ e volta no seu estado de repouso em t_{final}.

4.2.2 Velocidade Angular da Carga

A velocidade angular da carga está representada pela função $\phi'(t)$, medida em graus por segundos($\frac{°}{s}$).

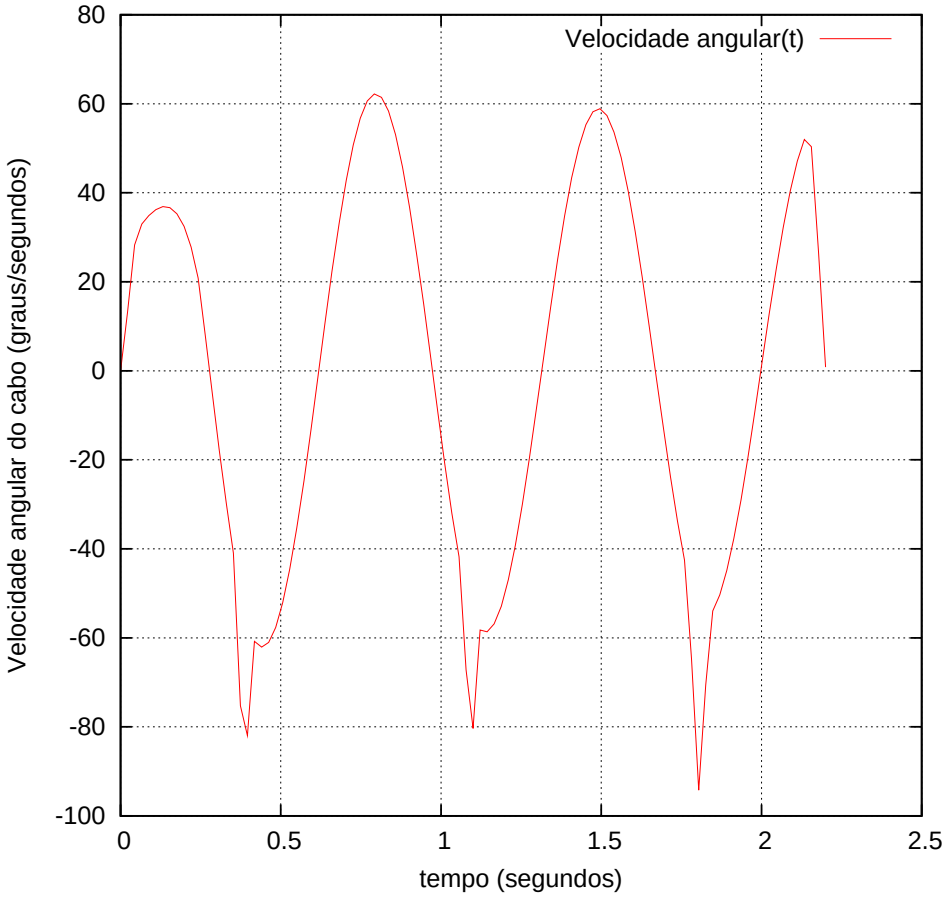

Figura 5: Velocidade angular da carga $t \times \phi'(t)$

Pelo gráfico (5), podemos notar que a carga sai de seu estado em repouso, oscila com uma velocidade entre $\frac{-100°}{s}$ e $\frac{70°}{s}$ e volta no seu estado de repouso em t_{final}

com velocidade nula.

4.2.3 Posição do Carro

A posição do carro está representada pela função $x_1(t)$, medida em metros (m).

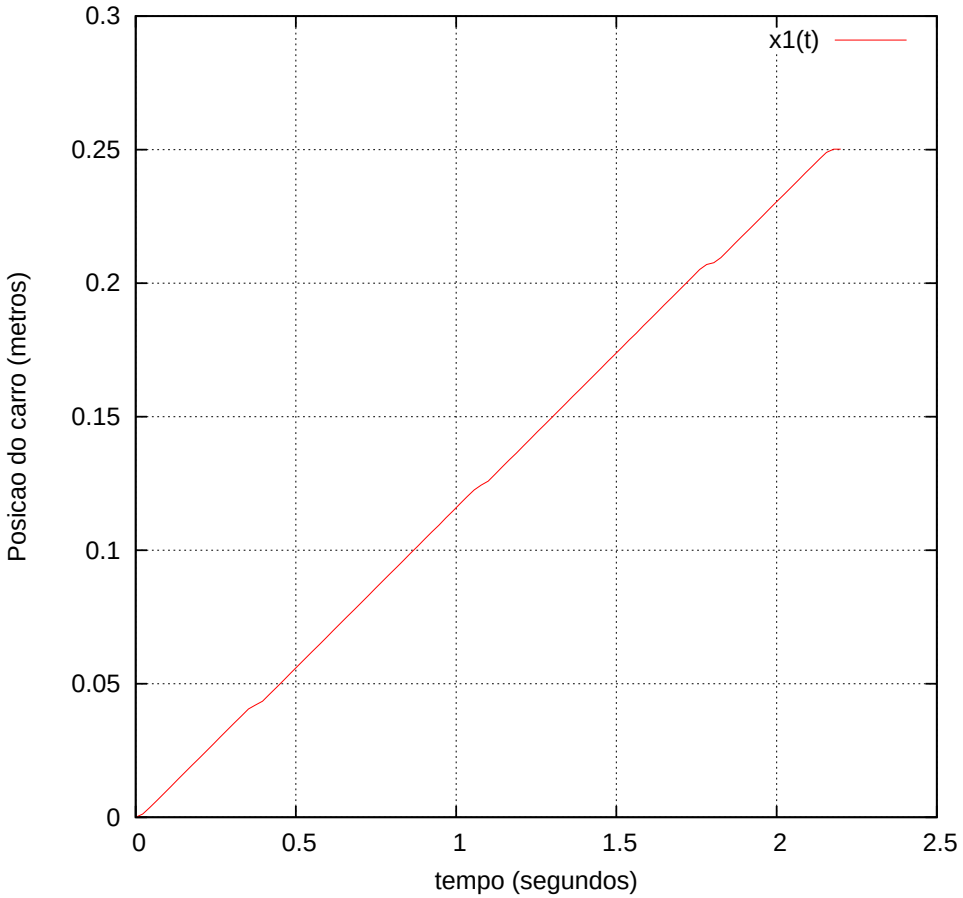

Figura 6: Posição do carro $t \times x_1(t)$

Pelo gráfico (6), podemos notar que carro sai de seu estado em repouso, e chega ao seu destino $(0, 25m)$ no tempo t mínimo, como desejado.

4.2.4 Velocidade do Carro

A velocidade do carro está representada pela função $x_1'(t)$, medida em metros por segundo ($\frac{m}{s}$).

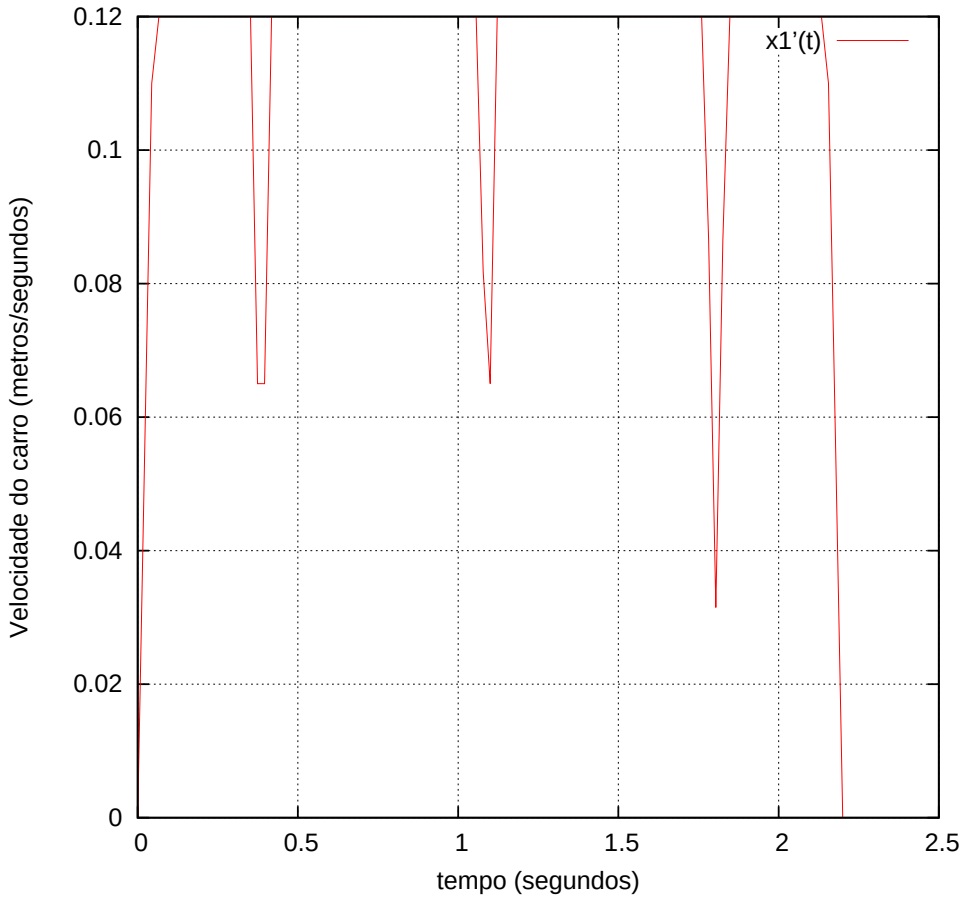

Figura 7: Velocidade do carro $t \times x_1'(t)$

Pelo gráfico (7), podemos notar que carro sai de seu estado em repouso, com velocidade máxima de $\frac{0{,}12m}{s}$ e chega ao seu destino com velocidade nula.

4.2.5 Aceleração do Carro

A aceleração do carro está representada pela função $x_1''(t)$, medida em metros por segundo ao quadrado ($\frac{m}{s^2}$).

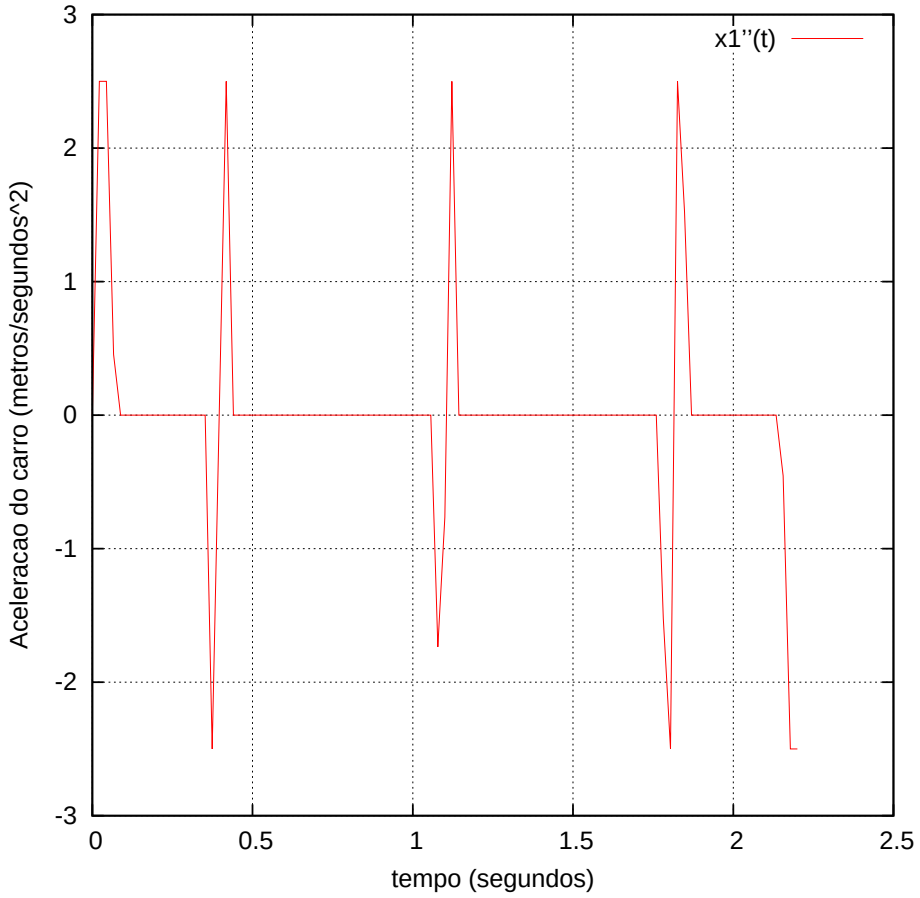

Figura 8: Aceleração do carro $t \times x_1''(t)$

Pelo gráfico (8), podemos notar que carro sai de seu estado em repouso, com aceleração mínima de $\frac{-2{,}5m}{s^2}$ e máxima de $\frac{2{,}5m}{s^2}$.

4.3 Otimização pelo GLPK

A seguir, mostraremos os gráficos de cada função relacionada aos três graus de liberdade $(\phi(t), \phi'(t), x_1(t), x_1'(t), x_1''(t))$ do nosso sistema mecânico.

4.3.1 Posição Angular da Carga

A posição angular da carga está representada pela função $\phi(t)$, medida em graus (°).

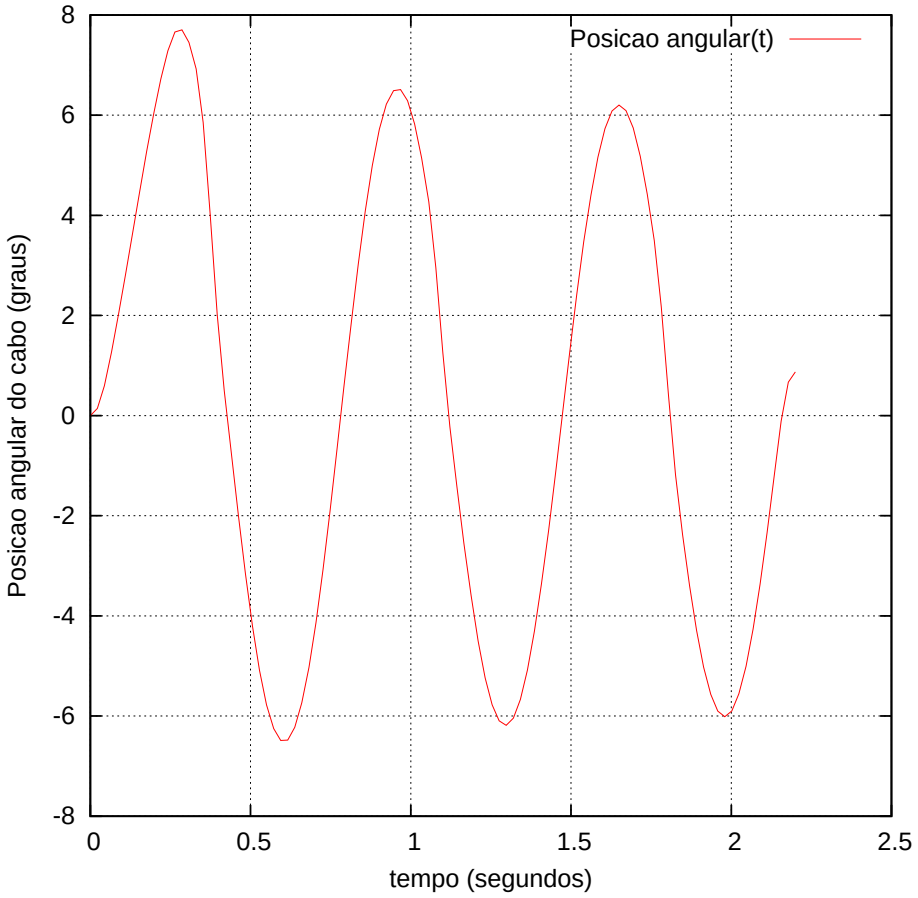

Figura 9: Posição angular da carga $t \times \phi(t)$

Pelo gráfico (9), podemos notar que a carga sai de seu estado em repouso, oscila entre $-8°$ e $8°$ e volta passando um pouco aproximadamente $1°$ do seu estado de repouso em t_{final}.

4.3.2 Velocidade Angular da Carga

A velocidade angular da carga está representada pela função $\phi'(t)$, medida em graus por segundos($\frac{°}{s}$).

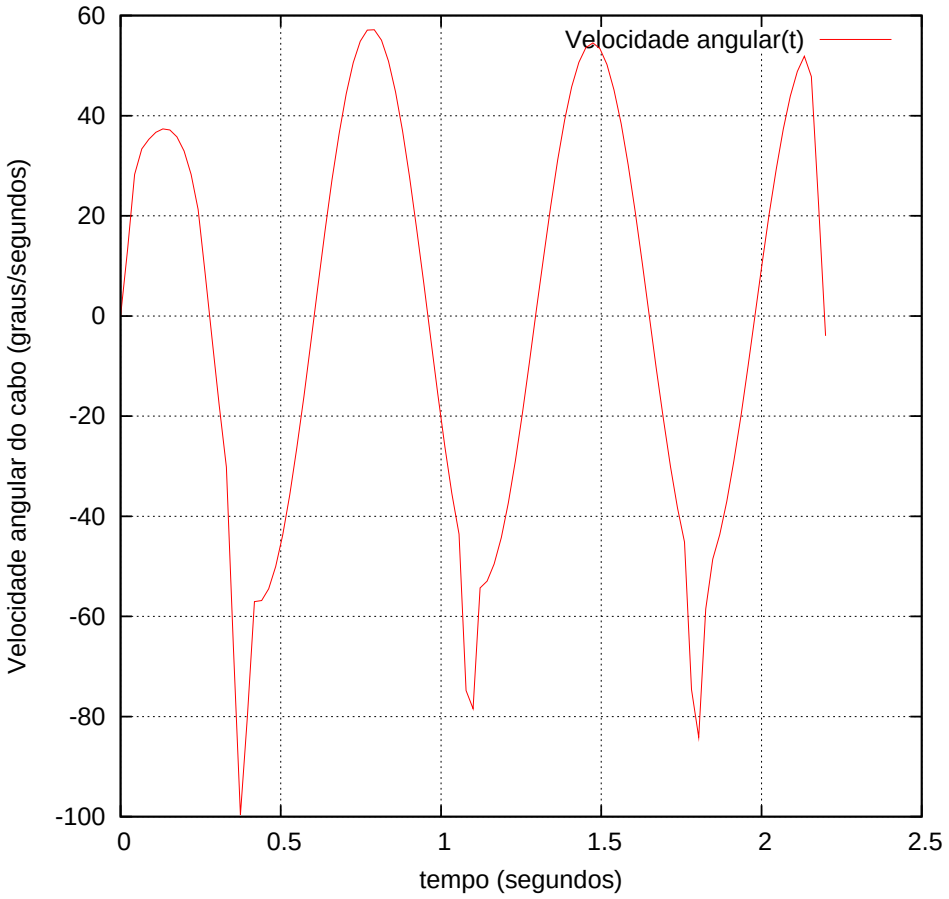

Figura 10: Velocidade angular da carga $t \times \phi'(t)$

Pelo gráfico (10), podemos notar que a carga sai de seu estado em repouso,

oscila com uma velocidade entre $\frac{-100^\circ}{s}$ e $\frac{60^\circ}{s}$ e passa um pouco seu estado de repouso em t_{final}.

4.3.3 Posição do Carro

A posição do carro está representada pela função $x_1(t)$, medida em metros (m).

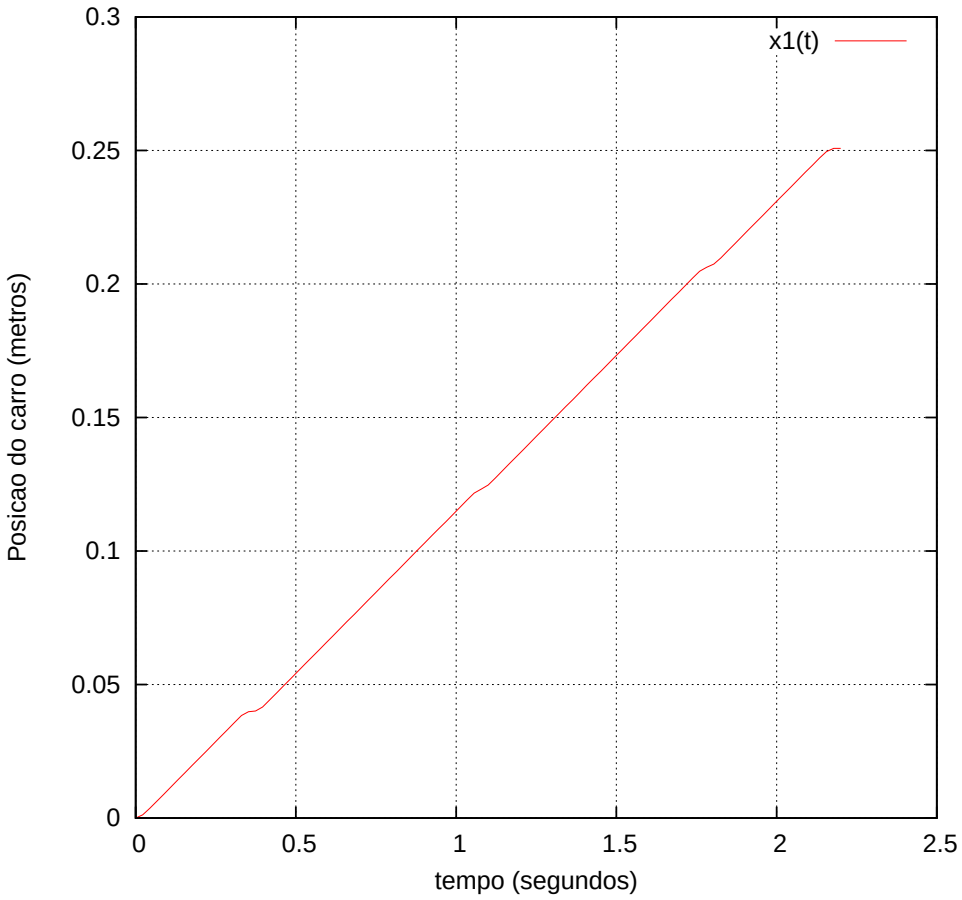

Figura 11: Posição do carro $t \times x_1(t)$

Pelo gráfico (11), podemos notar que carro sai de seu estado em repouso, e chega ao seu destino $(0,25m)$ no tempo t mínimo, como desejado.

4.3.4 Velocidade do Carro

A velocidade do carro está representada pela função $x_1'(t)$, medida em metros por segundo ($\frac{m}{s}$).

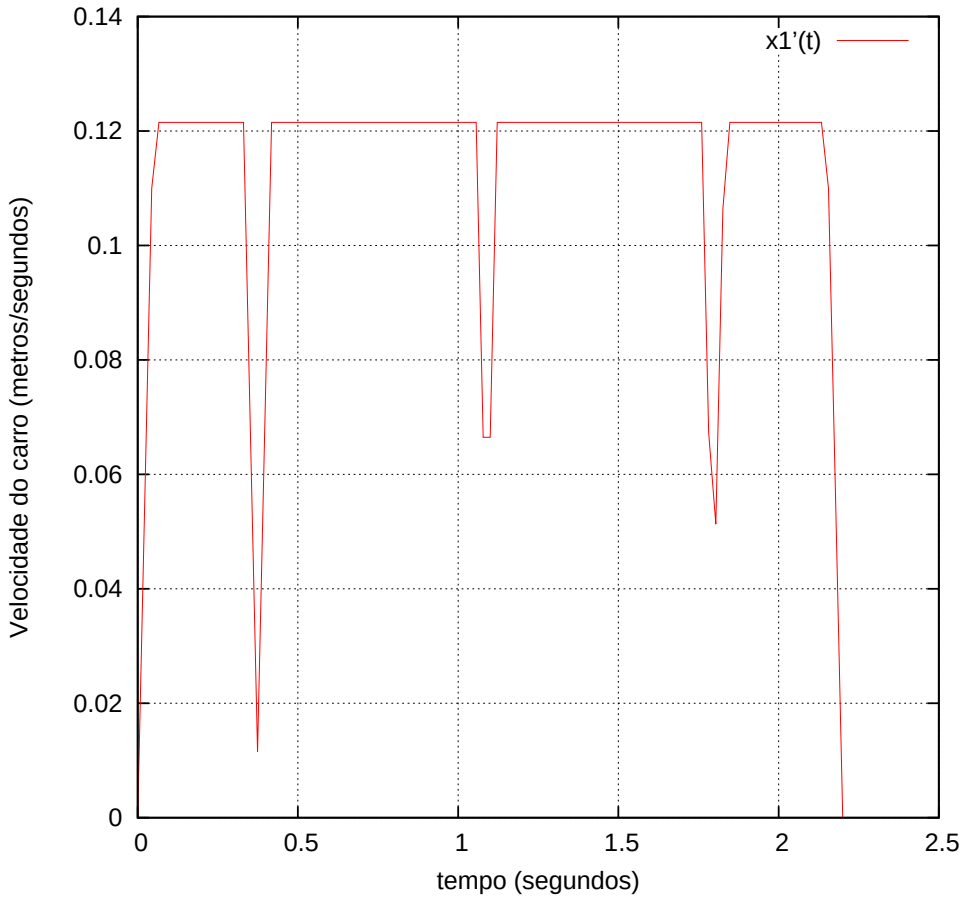

Figura 12: Velocidade do carro $t \times x_1'(t)$

Pelo gráfico (12), podemos notar que carro sai de seu estado em repouso, com velocidade máxima de aproximadamente $\frac{0{,}12m}{s}$ e chega ao seu destino com velocidade nula.

4.3.5 Aceleração do Carro

A aceleração do carro está representada pela função $x_1''(t)$, medida em metros por segundo ao quadrado ($\frac{m}{s^2}$).

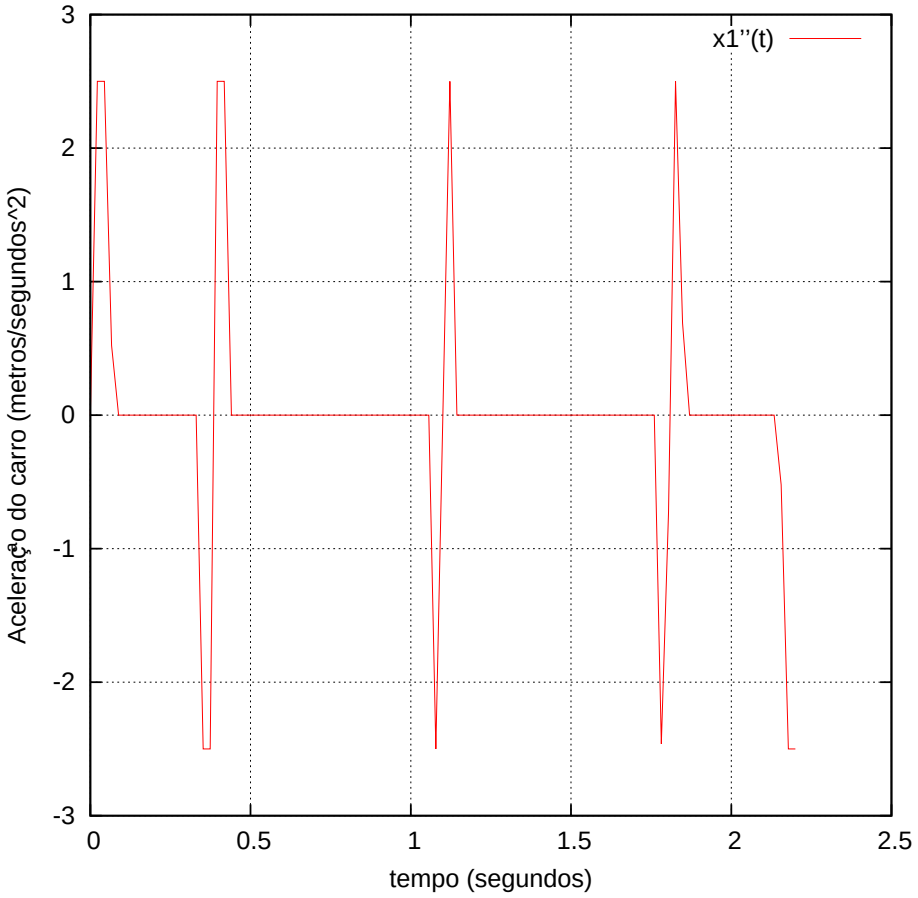

Figura 13: Aceleração do carro $t \times x_1''(t)$

Pelo gráfico (13), podemos notar que carro sai de seu estado em repouso, com aceleração mínima de $\frac{-2{,}5m}{s^2}$ e máxima de $\frac{2{,}5m}{s^2}$.

5 Conclusões

O projeto de controle ótimo para o nosso problema foi concluído com sucesso. Todos os resultados dos PPL's calculados da forma mais precisa (pelo Matlab) foram satisfatórios por atenderem as restrições e condições impostas.

Uma outra forma de afirmar tal sucesso, é ter analisado o trabalho experimental do mesmo problema que está sendo feito por um Professor de outra Universidade. Com os mesmos parâmetros, os resultados teóricos (deste trabalho) e os resultados experimentais são muito próximos entre si.

A forma tradicional de resolver o problema considerado neste trabalho seria com o uso do Princípio do Mínimo de Pontryagin. No entanto, neste caso, resultaria um conjunto de equações diferenciais não lineares com condições no contorno que não seriam triviais de resolver. A formulação do problema de tempo mínimo como uma seqüência de problemas de alcance máximo e a sua subseqüente discretização no tempo permitiu escrever o problema na forma de PPLs, ponto central do presente trabalho, dada a facilidade de obtenção de suas soluções numéricas.

6 Comentários

Não foram necessários os usos do *Método de Runge-Kutta de ordem quatro* ou da *Regra Composta de Simpson*, métodos mais poderosos por serem mais precisos que o utilizado no trabalho, apesar de terem sido testados também. Seus resultados foram satisfatoriamente iguais aos usados. Um estudo à parte foi feito, para a validação do método com problemas e soluções conhecidas, em que a ordem do Método do Ponto Médio sempre convergia para 2.

A biblioteca do Software livre GLPK foi adotado desde o início da otimização por isso não foi descartado quando foi descoberto quase no final do trabalho que sua precisão não estava boa. Assim, aproveitou-se a situação para mostrar a eficiência dos dois softwares utilizados. Não se descarta também a possibilidade de aumentar sua precisão.

Este trabalho é de caráter multidisciplinar por envolver áreas como Matemática Aplicada, Sistemas e Controle e Computação com aproveitamento do conhecimento de disciplinas tais como Análise Numérica, Métodos Numéricos em Equações Diferenciais, Controle, Programação Linear entre outras.

7 Apêndices

7.1 Apêndice A - Função λ

Dados $l_0 = 0,250m$, $l_f = 0,125m$ e $v_2 = 0,5m/s$ o comprimento inicial do cabo, o comprimento final do cabo e sua velocidade respectivamente, podemos obter $\lambda(\tau)$.

$$l(t) = l_0 + \frac{(l_f - l_0)}{(t_f - t_0)}t \tag{34}$$

$$l(t) = 0,25 - 0,5t \tag{35}$$

$$\lambda(t) = \frac{l(t)}{l_{min}} = 2 - 4t \Rightarrow \lambda(\tau) = 2 - 4\frac{\tau}{w_{max}} \tag{36}$$

onde $\tau = \omega_{max}t$, $\omega_{max} = \sqrt{\frac{g}{l_{min}}}$

O gráfico (14) do comprimento do cabo mostra a função $l(t)$.

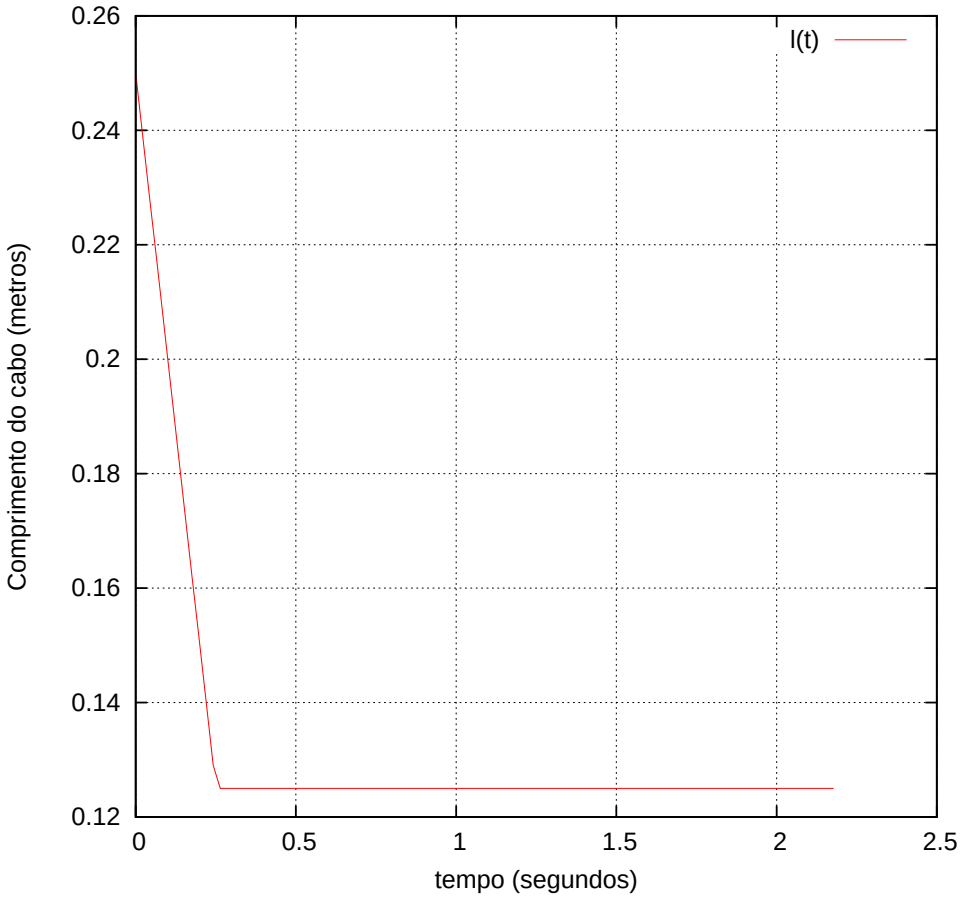

Figura 14: Comprimento do cabo $t \times l(t)$

7.2 Apêndice B - Problemas de Programação Linear

Problemas de Programação Linear (PPL) são problemas de otimização nos quais a função objetivo e as restrições são todas lineares. Estes problemas são do tipo

$$\begin{aligned} \text{maximizar} \quad & f(x) = \sum_{i=1}^{n} a_i x_i \\ \text{suj. a} \quad & \mathbf{Ax} \leqslant \mathbf{b} \\ & x_i \geqslant 0 \text{ (i = 1, ..., n)} \end{aligned} \tag{37}$$

onde $f(x)$ é a função objetivo sujeita às restrições $Ax \leqslant b$ e $x_i \geqslant 0$, sendo $\mathbf{A}$ uma matriz $m \times n$ e $\mathbf{x}$ e $\mathbf{b}$ vetores $n \times 1$.

Geometricamente, as restrições lineares definem um poliedro convexo, que é chamado de conjunto dos pontos viáveis. Uma vez que a função objetivo é também linear, todo ótimo local é automaticamente um ótimo global. A função objetivo ser linear também implica que uma solução ótima pode apenas ocorrer em um ponto da fronteira do conjunto de pontos viáveis.

Existem duas situações nas quais uma solução ótima não pode ser encontrada. Primeiro, se as restrições se contradizem (por exemplo, $x \geqslant 2$ e $x \leqslant 1$) logo, a região factível é vazia e não pode haver solução ótima, já que não pode haver solução nenhuma. Neste caso, o PPL é dito inviável.

Alternativamente, o poliedro pode ser ilimitado na direção da função objetivo (por exemplo: maximizar $x_1 + 3x_2$ sujeito a $x_1 \geqslant 0$, $x_2 \geqslant 0$, $x_1 + x_2 \geqslant 10$), neste caso não existe solução ótima uma vez que soluções arbitrariamente grandes da função objetivo podem ser construídas, e o problema é dito ilimitado.

Fora estas duas condições patológicas (que são freqüentemente eliminadas por limitações dos recursos inerentes ao problema que está sendo modelado, como acima), o ótimo é sempre alcançado num vértice do poliedro. Entretanto, o ótimo

nem sempre é único: é possivel ter um conjunto de soluções ótimas cobrindo uma aresta ou face do poliedro, ou até mesmo o poliedro todo (Esta última situação pode ocorrer se a função objetivo for uniformemente igual a zero).

7.3 Apêndice C - Integração Numérica

O método utilizado (ver [Burden, p.164]), está baseado na integral do polinômio interpolador de Lagrange num conjunto de nós distintos $x_0, ..., x_n$ do intervalo [a,b]

$$\int_a^b P_n(x) = \int_a^b \sum_{i=0}^{n} f(x_i) L_i(x) \tag{38}$$

somado com o erro de truncamento

$$E(f) = \frac{1}{(n+1)!} \int_a^b \prod_{i=0}^{n} (x - x_i) f^{n+1}(\xi(x)) dx \tag{39}$$

para obter $\int_a^b f(x)dx$. E portanto, calculando esta soma, concluimos que

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i) \tag{40}$$

é a aproximação da integral referida, onde $a_i = \int_a^b L_i(x)dx$ para cada $i = 0, 1, ..., n$.

No nosso caso, o método se restringe à **Regra de Simpson**, onde temos que

$$\begin{aligned} \int_{x_0}^{x_2} f(x)dx &= \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90} f^{(4)}(\xi) \\ &\approx \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] \end{aligned} \tag{41}$$

7.4 Apêndice D - Método de Runge-Kutta (Ponto Médio) para Equações de Ordem Superior e Sistemas de Equações Diferenciais

Os métodos de Runge-Kutta [7] possuem erro local de truncamento de alta ordem, como os métodos de Taylor, mas permitem se prescindir do cálculo e da avaliação das derivadas de f(t,y). Antes de apresentar as idéias por trás de suas derivações devemos enunciar o Teorema de Taylor para duas variáveis. A demonstração desse teorema pode ser obtida em qualquer livro de cálculo avançado.

Resumidamente (ver [Burden, p.237]), o primeiro passo na derivação do método de Runge-Kutta é determinar os valores de a_1, α_1 e β_1 com a propriedade de que $a_1 f(t+\alpha_1, y+\beta_1)$ aproxima

$$T^{(2)}(t,y) = f(t,y) + \frac{h}{2} f'(t,y) \tag{42}$$

com um erro não maior que $O(h^2)$, ou seja, o erro local de truncamento do método de Taylor de ordem dois.

O método da equação de diferenças que resulta da substituição de $T^{(2)}(t,y)$ por $f(t+(h/2), y+(h/2)f(t,y))$ no método de Taylor de ordem dois é um método específico de Runge-Kutta, conhecido como **Método do Ponto Médio** (utilizado neste trabalho de conclusão de curso):

$$\begin{aligned} w_0 &= \alpha, \\ w_{i+1} &= w_i + hf(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i)), \text{ para cada i = 0, 1, ..., N-1.} \end{aligned} \tag{43}$$

onde w_0 é a condição inicial da equação diferencial e h é o tamanho do passo.

No nosso caso, a equação em questão é de ordem superior e então tranformada em um sistema de equações difenciais de primeira ordem.

Um sistema de ordem m de problemas de valor inicial de primeira ordem pode ser expresso como

$$\begin{aligned} \frac{du_1}{dt} &= f_1(t, u_1, u_2, ..., u_m), \\ \frac{du_2}{dt} &= f_2(t, u_1, u_2, ..., u_m), \\ &... \\ \frac{du_m}{dt} &= f_m(t, u_1, u_2, ..., u_m), \end{aligned} \tag{44}$$

para $a \leqslant t \leqslant b$ com as condições iniciais

$$u_1(a) = \alpha_1, u_2(a) = \alpha_2, ..., u_m(a) = \alpha_m. \tag{45}$$

O objetivo é encontrar m funções $u_1, u_2, ..., u_m$ que satisfaçam o sistema de equações diferenciais e, também, todas as condições iniciais.

Para explicar a existência e a unicidade das soluções dos sistemas de equações é necessário estender a definição da condição de Lipschitz para as funções de algumas variáveis (ver [Burden, p.272]).

Os métodos para se resolver as equações diferenciais de primeira ordem são generalizações dos métodos para uma equação de primeira ordem. No nosso caso, o Método de Ruge-Kutta de ordem dois (Ponto Médio) é generalizado como

$$\begin{aligned} w_{i,j} &= \alpha, \text{para i} = 0, 1 \text{ e j} = 0, 1 \\ w_{i+1,j+1} &= w_{i,j} + hf(t_i + \frac{h}{2}, w_{i,j} + \frac{h}{2} f(t_i, w_{i,j})), \\ &\quad \text{para cada i} = 0, 1, ..., \text{m-1 e para cada j} = 0, 1, ..., \text{N-1.} \end{aligned} \tag{46}$$

7.5 Apêndice E -Comparação entre os valores de u's

A primeira coluna é composta pelo número do passo, a segunda pelos u's otimizados pelo GLPK e a terceira pelos u's otimizados pelo Matlab. Podemos observar que a partir do terceiro passo, já há divergências de arredondamento.

```
1 1.0000000e+00 1.0000000e+000
2 1.0000000e+00 1.0000000e+000
3 1.8181818e-01 1.8181818e-001
4 0.0000000e+00 1.1036055e-016
5 0.0000000e+00 -5.1367190e-017
6 0.0000000e+00 2.7046598e-016
7 0.0000000e+00 -1.6887102e-016
8 0.0000000e+00 -8.3278480e-018
9 0.0000000e+00 -2.2272039e-016
10 0.0000000e+00 2.1657464e-016
11 0.0000000e+00 1.3216736e-016
12 0.0000000e+00 4.5168622e-017
13 8.6472584e-16 -2.4268326e-016
14 -3.1398063e-16 -8.2401257e-016
15 6.8517263e-17 5.7432116e-016
16 -1.0000000e+00 1.6766912e-016
17 5.1398772e-15 -1.0000000e+000
18 1.0000000e+00 -2.0995921e-016
19 -1.0585806e-15 1.0000000e+000
20 1.7770074e-16 5.3731289e-016
21 -6.8149454e-17 4.4034283e-016
22 6.8149454e-17 -1.6532459e-016
23 0.0000000e+00 -1.0004076e-016
```

```
24 0.0000000e+00 -4.5454445e-016
25 0.0000000e+00 2.6730195e-016
26 0.0000000e+00 3.5261365e-016
27 0.0000000e+00 4.2602733e-016
28 0.0000000e+00 -3.9184812e-016
29 0.0000000e+00 2.4333731e-016
30 0.0000000e+00 -9.9392282e-016
31 0.0000000e+00 6.2505129e-016
32 0.0000000e+00 4.5726451e-016
33 0.0000000e+00 -1.0414613e-015
34 0.0000000e+00 5.9944951e-016
35 0.0000000e+00 4.9981601e-016
36 0.0000000e+00 -2.1124424e-016
37 0.0000000e+00 8.0919363e-016
38 0.0000000e+00 -7.3611212e-016
39 0.0000000e+00 -3.2017282e-016
40 0.0000000e+00 -5.6563175e-017
41 0.0000000e+00 -1.7275813e-015
42 0.0000000e+00 7.2013309e-016
43 0.0000000e+00 1.2848414e-015
44 0.0000000e+00 -1.3644485e-016
45 0.0000000e+00 1.7864365e-016
46 -1.3936823e-16 -2.6850055e-016
47 1.3936823e-16 -5.0964815e-016
48 -1.0000000e+00 -4.7947840e-016
49 -9.5000564e-01 -6.9431872e-001
50 1.0000000e+00 -3.0568128e-001
```

51 9.5000564e-01 1.0000000e+000

52 0.0000000e+00 -4.8527675e-016

53 1.9740490e-16 2.5536042e-016

54 6.5888794e-16 1.5057937e-015

55 -8.5629284e-16 -6.6944958e-016

56 0.0000000e+00 9.0462787e-016

57 0.0000000e+00 -2.2682426e-016

58 0.0000000e+00 -5.8990694e-016

59 0.0000000e+00 9.3755477e-017

60 0.0000000e+00 7.3672828e-016

61 0.0000000e+00 2.6141360e-016

62 0.0000000e+00 -1.6585890e-016

63 0.0000000e+00 -1.7354081e-015

64 0.0000000e+00 1.6707466e-015

65 0.0000000e+00 -9.0874638e-016

66 0.0000000e+00 6.9608299e-016

67 0.0000000e+00 3.8695096e-016

68 0.0000000e+00 -6.0244492e-016

69 3.2304151e-16 3.5152955e-016

70 -7.6509563e-16 3.4204233e-016

71 4.4205412e-16 -1.2200260e-015

72 9.6535882e-16 8.4408636e-016

73 -9.6535882e-16 -2.1136532e-016

74 0.0000000e+00 4.7356341e-016

75 3.8220354e-16 3.9686547e-016

76 6.6870192e-16 -1.7431807e-015

77 -1.0509055e-15 -4.7069428e-017

78 0.0000000e+00 -1.2016524e-015

79 0.0000000e+00 -6.4083063e-016

80 -8.7518788e-03 1.0435312e-015

81 -1.0000000e+00 -6.0925401e-001

82 1.0000000e+00 -1.0000000e+000

83 8.7518788e-03 1.0000000e+000

84 4.3839036e-15 6.0925401e-001

85 -2.5914392e-15 -3.2712905e-016

86 -2.8193191e-15 2.3360294e-016

87 1.7620759e-15 1.2708605e-015

88 0.0000000e+00 -1.1185948e-015

89 0.0000000e+00 2.3282391e-016

90 0.0000000e+00 1.3041876e-015

91 0.0000000e+00 5.1143348e-016

92 0.0000000e+00 -1.4448085e-016

93 0.0000000e+00 7.1679923e-016

94 0.0000000e+00 -8.4982463e-016

95 0.0000000e+00 -1.2733931e-015

96 0.0000000e+00 3.2970699e-016

97 0.0000000e+00 1.0311707e-015

98 -1.8181818e-01 -1.8181818e-001

99 -1.0000000e+00 -1.0000000e+000

100 -1.0000000e+00 -1.0000000e+000

7.6 Apêndice F - Programa fonte do Simulador

```
/* Comentarios:
* Esse programa resolve as equacoes de estado
* O programa gera saidas (arquivos) para cada variavel de estudo.
* No gnuplot, digite <plot <arquivo> w l>, para ver a resolucao
do problema
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define A 0
#define B 2.2 //em t, tau = t*wmax
#define N 100
#define T0 0
#define ACMAX 2.5
#define WMAX 8.8
#define G 9.8
#define PI 3.14
#define LMIN 0.125
#define LMAX 0.25

//sistema das equacoes diferenciais
void funcao(double tau, double **wk, double **f);

//metodo do ponto medio
void ponto_medio(double **wk, double **c, double **f, double
```

```
**k, double h, double tk );

//integracao numerica
void integracao(double x0, double x2, double *f_int_x0, double
*f_int_x1, double *f_int_x2, double *resultado);

//calcula a inversa da matriz
void inv_matriz(double **matriz, double **inversa);

//calcula o determinante da matriz 2x2
double determinante_2x2(double **matriz);

//funcao lambda do sistema
double lambda(double tau);

//derivada da funcao lambda
double derivada_lambda(double tau);

//funcao entrada u1
double funcao_entrada(double t);

//multiplicacao de uma matriz 2x2 por uma 2x1 (vetor tam 2)
void mult_2x2(double **matriz, double *vetor, double *resultado);

//soma 2 vetores
void soma_vetor(double *vetor1, double *vetor2, double *resultado);
```

```
//atribui o valor zero a cada posicao do vetor
void zera_vetor(double *vetor);

//atribui valor zero a cada posicao da matriz
void zera_matriz(double **matriz);

//transforma a matriz em uma matriz unitaria
void matriz_unitaria(double **matriz);

int main (){
int i, //iteracoes
opcao,
tempo_u[100], //tempo entrada
n = N; //numero de iteracoes

double a = A, b = B*WMAX, //intervalo de A a B
to = T0, //tempo inicial
h, //passo das iteracoes
tk, //tempo tau
t, //tempo t
u[100], //entrada otimizada
acmax = ACMAX, //aceleracao maxima do carro
wmax = WMAX, //raiz de g/lmin
g = G, //aceleracao da gravidade
pi = PI,
x1[100], //posicao do carro
vel1[100]; //velocidade do carro
```

```
//matrizes auxiliares do metodo
double **c, **f, **k;

//matrizes que recebem os valores de phi
double **wk, **wk1;

//(vetor b) * u
double Bu[2];

//recebe inversa de phi
double **inv_fi;

//recebe inversa(phi) * b * u
double fi_Bu_x0[2], fi_Bu_x1[2], fi_Bu_x2[2];

//recebe o resultado da integral
double integrado[2];

//recebe phi * x(k)
double fi_x1[2], fi_x2[2];

//recebe x(k+1) da iteracao anterior
double xk0[2], xk[2], phi[2];

//saida em arquivo
FILE *saida_l,
```

```
*saida_pos_ang,
*saida_vel_ang,
*saida_x1,
*saida_vel1,
*saida_a1,
*entrada;

//locacao de memoria para as matrizes
c = (double**) malloc((2)*sizeof(double*));
f = (double**) malloc((2)*sizeof(double*));
k = (double**) malloc((2)*sizeof(double*));
wk = (double**) malloc((2)*sizeof(double*));
wk1 = (double**) malloc((2)*sizeof(double*));

inv_fi = (double**) malloc((2)*sizeof(double*));

if (!c || !f || !k || !wk || !wk1 || !inv_fi) {
fprintf(stderr, "** Erro:  Memoria Insuficiente **");
exit(1);
}

for (i=0; i<2; i++) {
c[i] = (double*) malloc((2)*sizeof(double));
f[i] = (double*) malloc((2)*sizeof(double));
k[i] = (double*) malloc((2)*sizeof(double));
wk[i] = (double*) malloc((2)*sizeof(double));
wk1[i] = (double*) malloc((2)*sizeof(double));
```

```
inv_fi[i] = (double*) malloc((2)*sizeof(double));

if (!c[i] || !f[i] || !k[i] || !wk[i] || !wk1[i] || !inv_fi[i])
{
fprintf(stderr, "** Erro:  Memoria Insuficiente **");
exit(2);
}
}

//phi inicial = matriz identidade
matriz_unitaria(wk);
matriz_unitaria(wk1);

//(b * u)[0] = 0 e (b * u)[1] = u/lambda (adiante)
zera_vetor(Bu);

//condicao inicial
zera_vetor(xk0);

h = (b-a)/n;
tk = to;

printf("Digite 1 para a resolucao do problema otimizado pelo
glpk, 2 pelo matlab ou 3 para as simulacoes.  'ń");
scanf("%d", &opcao);
```

```
if(opcao == 1){
if(!(entrada = fopen("resposta_u", "r")))
exit(1);

for(i=1; i<=100; i++)
fscanf(entrada, "%d %lf", &tempo_u[i], &u[i]);

fclose(entrada);
}

if(opcao == 2){
if(!(entrada = fopen("u", "r")))
exit(1);

for(i=1; i<=100; i++)
fscanf(entrada, "%lf", &u[i]);

fclose(entrada);
}

printf("Favor confirmar o numero ’ń");
scanf("%d", &opcao);

if(!(saida_l = fopen("saida_l.txt", "w")))
exit(1);
if(!(saida_pos_ang = fopen("saida_pos_ang.txt", "w")))
exit(1);
```

```
if(!(saida_vel_ang = fopen("saida_vel_ang.txt", "w")))
exit(1);
if(!(saida_x1 = fopen("saida_pos_carro.txt", "w")))
exit(1);
if(!(saida_vel1 = fopen("saida_vel_carro.txt", "w")))
exit(1);
if(!(saida_a1 = fopen("saida_acel_carro.txt", "w")))
exit(1);

zera_matriz(c);
zera_matriz(f);
zera_matriz(k);

x1[0] = 0;
vel1[0] = 0;

//imprime o vetor no arquivo
fprintf(saida_pos_ang, "%.3e %.3e’;;;;ń", 0.0, 0.0);

//imprime o vetor no arquivo
fprintf(saida_vel_ang, "%.3e %.3e’;;;;ń", 0.0, 0.0);

//imprime o vetor no arquivo
fprintf(saida_a1, "%.3e %.3e’;;;;ń", 0.0, 0.0);

//imprime o vetor no arquivo
fprintf(saida_x1, "%.3e %.3e’;;;;ń", 0.0, 0.0);
```

```
//imprime o vetor no arquivo
fprintf(saida_vel1, "%.3e %.3e’;;;;ń", 0.0, 0.0);

for(i=1; i<=n; i++){

//funcao fi (matriz)
ponto_medio(wk, c, f, k, h, tk);

inv_matriz(wk1, inv_fi);
if((opcao == 1) || (opcao == 2))
Bu[1] = u[i]/lambda(tk);//x0
else
Bu[1] = funcao_entrada(tk)/lambda(tk);//x0
mult_2x2(inv_fi, Bu, fi_Bu_x0);

ponto_medio(wk1, c, f, k, h/2.0, tk);

inv_matriz(wk1, inv_fi);
if((opcao == 1) || (opcao == 2))
Bu[1] = u[i]/lambda(tk+(h/2.0));//x1
else
Bu[1] = funcao_entrada(tk+(h/2.0))/lambda(tk+(h/2.0));//x1
mult_2x2(inv_fi, Bu, fi_Bu_x1);

ponto_medio(wk1, c, f, k, h/2.0, tk+(h/2.0));
```

```
inv_matriz(wk1, inv_fi);
if((opcao == 1) || (opcao == 2))
Bu[1] = u[i]/lambda(tk+h);//x2
else
Bu[1] = funcao_entrada(tk+h)/lambda(tk+h);//x2
mult_2x2(inv_fi, Bu, fi_Bu_x2);

//integracao do vetor fi*Bu no intervalo k -> tk-h a k+1 ->
tk
integracao (tk, tk+h, fi_Bu_x0, fi_Bu_x1, fi_Bu_x2, integrado);

//fi * x(k)
mult_2x2(wk, xk0, fi_x1);

//fi * integral(inversa(fi)*Bu)
mult_2x2(wk1, integrado, fi_x2);

//resultado final do vetor x(k+1) = fi * x(k) + fi * integral(inversa(fi)*B*u)
soma_vetor(fi_x1, fi_x2, xk);

//conversao dimensionalizada em graus
phi[0] = (xk[0] * 180.0 * acmax)/(g * pi);
phi[1] = (xk[1] * 180.0 * acmax * wmax)/(g * pi);

if((opcao == 1) || (opcao == 2)){
vel1[i] = vel1[i-1] + h*u[i]*acmax/wmax;
x1[i] = x1[i-1] + (h*vel1[i])/wmax;
```

```
}
else{
vel1[i] = vel1[i-1] + h*funcao_entrada(tk+(h/2.0))*acmax/wmax;
x1[i] = x1[i-1] + (h*funcao_entrada(tk+(h/2.0)))/wmax;
}

//imprime o vetor no arquivo
fprintf(saida_l, "%.3e %.3e’ýýýýń", tk/wmax, lambda(tk)*LMIN);

//tk passa pra proxima iteracao
tk = tk + h;

t = tk/wmax;

//imprime o vetor no arquivo
fprintf(saida_pos_ang, "%.3e %.3e’ýýýýń", t, phi[0]);

//imprime o vetor no arquivo
fprintf(saida_vel_ang, "%.3e %.3e’ýýýýń", t, phi[1]);

//imprime o vetor no arquivo
if((opcao == 1) || (opcao == 2))
fprintf(saida_a1, "%.3e %.3e’ýýýýń", t, u[i]*acmax);
else
fprintf(saida_a1, "%.3e %.3e’ýýýýń", t, funcao_entrada(tk-(h/2.0)));

//imprime o vetor no arquivo
```

```
fprintf(saida_x1, "%.3e %.3e’;;;;ń", t, x1[i]);

//imprime o vetor no arquivo
fprintf(saida_vel1, "%.3e %.3e’;;;;ń", t, vel1[i]);

xk0[0] = xk[0];
xk0[1] = xk[1];

matriz_unitaria(wk);
}

fclose(saida_l);
fclose(saida_pos_ang);
fclose(saida_vel_ang);
fclose(saida_x1);
fclose(saida_vel1);
fclose(saida_a1);

return 0;
}

double lambda(double tau){
double res;
res = 2 - 4*tau/WMAX;
if(res <= 1)
return 1;
else if(res >= LMAX/LMIN)
```

```
return LMAX/LMIN;
else
return res;
}

double derivada_lambda(double tau){
if(tau <= (WMAX/4.0))
return (-4.0/WMAX);
else
return 0;
}

double funcao_entrada(double t){
if(t == 0) t = 1.0;
return 1.0/t;
}

void funcao(double tau, double **wk, double **f){
double lamb, lambl;
lamb = lambda(tau);
lambl = derivada_lambda(tau);

//vetor 1
f[0][0] = wk[1][0];
f[1][0] = (-wk[0][0]-(2.0*wk[1][0]*lambl))/lamb;

//vetor 2
```

```
f[0][1] = wk[1][1];
f[1][1] = (-wk[0][1]-(2.0*wk[1][1]*lambl))/lamb;
}

void ponto_medio(double **wk, double **c, double **f, double
**k, double h, double tk ){
funcao(tk, wk, c);
f[0][0] = wk[0][0] + (h/2.0)*c[0][0];
f[0][1] = wk[0][1] + (h/2.0)*c[0][1];
f[1][0] = wk[1][0] + (h/2.0)*c[1][0];
f[1][1] = wk[1][1] + (h/2.0)*c[1][1];

funcao(tk+(h/2.0), f, k);
wk[0][0] = wk[0][0] + h*k[0][0];
wk[0][1] = wk[0][1] + h*k[0][1];
wk[1][0] = wk[1][0] + h*k[1][0];
wk[1][1] = wk[1][1] + h*k[1][1];
}

void integracao(double x0, double x2, double *f_int_x0, double
*f_int_x1, double *f_int_x2, double *resultado){
int i;
double x1, h;
h = (x2-x0)/2.0;
x1 = (x2 + x0)/2.0;
for(i=0; i<2; i++)
resultado[i] = h*(f_int_x0[i]+4.0*f_int_x1[i]+f_int_x2[i])/3.0;
```

```
}

void inv_matriz(double **matriz, double **inversa){
double det;
det = determinante_2x2(matriz);
inversa[0][0] = matriz[1][1]/det;
inversa[0][1] = -matriz[0][1]/det;
inversa[1][0] = -matriz[1][0]/det;
inversa[1][1] = matriz[0][0]/det;
}

double determinante_2x2(double **matriz){
return matriz[0][0] * matriz[1][1] - matriz[0][1] * matriz[1][0];
}

void mult_2x2(double **matriz, double *vetor, double *resultado){
resultado[0] = matriz[0][0]*vetor[0] + matriz[0][1]*vetor[1];
resultado[1] = matriz[1][0]*vetor[0] + matriz[1][1]*vetor[1];
}

void soma_vetor(double *vetor1, double *vetor2, double *resultado){
int i;
for(i=0; i<2; i++)
resultado[i] = vetor1[i] + vetor2[i];
}

//tranforma em um vetor nulo
```

```
void zera_vetor(double *vetor){
int i;
for(i=0; i<2; i++)
vetor[i] = 0.0;
}

//tranforma em uma matriz nula
void zera_matriz(double **matriz){
int i, j;
for(i=0; i<2; i++)
for(j=0; j<2; j++)
matriz[i][j] = 0.0;
}

//tranforma em uma matriz unitaria
void matriz_unitaria(double **matriz){
matriz[0][0] = 1; matriz[0][1] = 0; matriz[1][0] = 0; matriz[1][1]
= 1;
}
```

7.7 Apêndice G - Programa Fonte do Otimizador

```
/* Comentarios:
* Esse programa resolve Problema de Programacao Linear (PPL)
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "glpk.h"
#define LMIN 0.125
#define LMAX 0.25
#define G 9.8
#define ATMAX 2.5
#define N 100
#define T0 0
#define A 0
#define B 2.2 //em t, (tau = 2.2 * wmax)
#define WMAX 8.8

//sistema das equacoes diferenciais
void funcao(double tau, double **wk, double **f);

//metodo do ponto medio
void ponto_medio(double **wk, double **c, double **f, double
**k, double h, double tk );

//integracao numerica
```

```
void integracao(double x0, double x2, double *f_int_x0, double
*f_int_x1, double *f_int_x2, double *resultado);

//calcula a inversa da matriz
void inv_matriz(double **matriz, double **inversa);

//calcula o determinante da matriz 2x2
double determinante_2x2(double **matriz);

//funcao lambda do sistema
double lambda(double tau);

//derivada da funcao lambda
double derivada_lambda(double tau);

//multiplicacao de uma matriz 2x2 por uma 2x1 (vetor tam 2)
void mult_2x2(double **matriz, double *vetor, double *resultado);

//atribui o valor zero a cada posicao do vetor
void zera_vetor(double *vetor);

//atribui valor zero a cada posicao da matriz
void zera_matriz(double **matriz);

//transforma a matriz em uma matriz unitaria
void matriz_unitaria(double **matriz);
```

```
int main (){
int i, j, //variaveis de iteracao
n = N; //iteracoes

double a = A, b = B*WMAX, //intervalo de a a b
to = T0, //tempo inicial
h, //passo
tk; //tempo

//matrizes auxiliares no metodo do ponto medio
double **c, **f, **k;

//matrizes que recebem o calculo do ponto medio
double **wk, **wk1;

//vetor b da equacao de estado
double Bb[2];

//recebe a matriz inversa
double **inv_fi;

//recebe inversa(phi) * b
double fi_B_x0[2], fi_B_x1[2], fi_B_x2[2];

//recebe o resultado da integracao numerica
double integrado[2];
```

```
//recebe phi * (resultado da integral)
double fi_x2[2];

FILE *saida_gama,
*saida_u;

//otimizacao
LPX *lp; //parse
int ia[1+10500],
ja[1+10500],
t; //iteracao

double matriz[1+1000][1+1000], //matriz do PPL
ar[1+10500], Z, u[1+100],
objetivo[1+100], //funcao objetivo
vel_max=0.12; //velocidade maxima do carro em funcao do tempo
t

c = (double**) malloc((2)*sizeof(double*));
f = (double**) malloc((2)*sizeof(double*));
k = (double**) malloc((2)*sizeof(double*));
wk = (double**) malloc((2)*sizeof(double*));
wk1 = (double**) malloc((2)*sizeof(double*));

inv_fi = (double**) malloc((2)*sizeof(double*));

if (!c || !f || !k || !wk || !wk1 || !inv_fi) {
```

```
fprintf(stderr, "** Erro:  Memoria Insuficiente **");
exit(1);
}

for (i=0; i<2; i++) {
c[i] = (double*) malloc((2)*sizeof(double));
f[i] = (double*) malloc((2)*sizeof(double));
k[i] = (double*) malloc((2)*sizeof(double));
wk[i] = (double*) malloc((2)*sizeof(double));
wk1[i] = (double*) malloc((2)*sizeof(double));
inv_fi[i] = (double*) malloc((2)*sizeof(double));

if (!c[i] || !f[i] || !k[i] || !wk[i] || !wk1[i] || !inv_fi[i])
{
fprintf(stderr, "** Erro:  Memoria Insuficiente **");
exit(2);
}
}

matriz_unitaria(wk);
matriz_unitaria(wk1);

zera_vetor(Bb);

h = (b-a)/n; //delta tal
tk = to;
```

```
if(!(saida_gama = fopen("resposta_gama", "w")))
exit(1);

zera_matriz(c);
zera_matriz(f);
zera_matriz(k);

for(i=0; i<n; i++){

//funcao fi (matriz)
ponto_medio(wk, c, f, k, h, tk);

inv_matriz(wk1, inv_fi);
Bb[1] = 1.0/lambda(tk);//x0
mult_2x2(inv_fi, Bb, fi_B_x0);

ponto_medio(wk1, c, f, k, h/2.0, tk);

inv_matriz(wk1, inv_fi);
Bb[1] = 1.0/lambda(tk+(h/2.0));//x1
mult_2x2(inv_fi, Bb, fi_B_x1);

ponto_medio(wk1, c, f, k, h/2.0, tk+(h/2.0));

inv_matriz(wk1, inv_fi);
Bb[1] = 1.0/lambda(tk+h);//x2
mult_2x2(inv_fi, Bb, fi_B_x2);
```

```
//integracao do vetor fi*B no intervalo k -> tk-h a k+1 -> tk
integracao (tk, tk+h, fi_B_x0, fi_B_x1, fi_B_x2, integrado);

//fi * integral(inversa(fi)*B)
mult_2x2(wk, integrado, fi_x2);

//imprime o vetor Gama no arquivo
fprintf(saida_gama, "%.3e %.3e'ń", fi_x2[0], fi_x2[1]);

//atribui o vetor Gama na matriz de otimizacao
matriz[1][i+1] = fi_x2[0];
matriz[2][i+1] = fi_x2[1];

tk = tk + h;
matriz_unitaria(wk);
}

fclose(saida_gama);

/*------------- OTIMIZACAO ---------------*/

if(!(saida_u = fopen("resposta_u", "w")))
exit(1);

for(i=1; i<=N; i++){
objetivo[i] = (h*h/2.0)*(2.0*(N - i) +1.0);
```

```
}

//linha 3 a (n-1 + 3) - n nao participa da somatoria!
//matriz A3 quadrada:
for(i=3; i<=N+1; i++)
for(j=1; j<=N; j++)
if(i-2>=j) //matriz triangular inferior a[i][j]=1
matriz[i][j] = 1.0;
else
matriz[i][j] = 0.0;

for(j=1; j<=N; j++){
matriz[N+2][j] = h;
}

lp = lpx_create_prob();
lpx_set_prob_name(lp, "otimizacao");
lpx_set_obj_dir(lp, LPX_MAX);

lpx_add_rows(lp, N+2);
lpx_set_row_bnds(lp, 1, LPX_FX, 0.0, 0.0);
lpx_set_row_bnds(lp, 2, LPX_FX, 0.0, 0.0);
for(i=3; i<=N+1; i++)
lpx_set_row_bnds(lp, i, LPX_DB, (-1*vel_max*G)/(h*ATMAX*LMIN*WMAX),
(vel_max*G)/(h*ATMAX*LMIN*WMAX));

lpx_set_row_bnds(lp, N+2, LPX_FX, 0.0, 0.0);
```

```
lpx_add_cols(lp, N);
for(i=1; i<=N; i++){
lpx_set_col_bnds(lp, i, LPX_DB, -1.0, 1.0);
lpx_set_obj_coef(lp, i, objetivo[i]);
}

t=1;
for(i=1; i<=N+2; i++){
for(j=1; j<=N; j++){
if(matriz[i][j] == 0.0)
continue;
ia[t]=i;
ja[t]=j;
ar[t] = matriz[i][j];
t=t+1;
}
}

lpx_load_matrix(lp, t-1, ia, ja, ar);
lpx_simplex(lp);
Z = lpx_get_obj_val(lp);

for(i=1; i<=N; i++){
u[i] = lpx_get_col_prim(lp, i);
fprintf(saida_u, "%d %.3e'ń", i, u[i]);
}
```

```
fclose(saida_u);

printf("Z = %g ’ń", Z);

lpx_delete_prob(lp);

return 0;
}

double lambda(double tau){
double res;
res = 2 - 4*tau/WMAX;
if(res <= 1)
return 1;
else if(res >= LMAX/LMIN)
return LMAX/LMIN;
else
return res;
}

double derivada_lambda(double tau){
if(tau <= (WMAX/4.0))
return (-4.0/WMAX);
else
return 0;
}
```

```
void funcao(double tau, double **wk, double **f){
double lamb, lambl;
lamb = lambda(tau);
lambl = derivada_lambda(tau);

//vetor 1
f[0][0] = wk[1][0];
f[1][0] = (-wk[0][0]-(2.0*wk[1][0]*lambl))/lamb;

//vetor 2
f[0][1] = wk[1][1];
f[1][1] = (-wk[0][1]-(2.0*wk[1][1]*lambl))/lamb;

}

void ponto_medio(double **wk, double **c, double **f, double
**k, double h, double tk ){
funcao(tk, wk, c);
f[0][0] = wk[0][0] + (h/2.0)*c[0][0];
f[0][1] = wk[0][1] + (h/2.0)*c[0][1];
f[1][0] = wk[1][0] + (h/2.0)*c[1][0];
f[1][1] = wk[1][1] + (h/2.0)*c[1][1];

funcao(tk+(h/2.0), f, k);
wk[0][0] = wk[0][0] + h*k[0][0];
wk[0][1] = wk[0][1] + h*k[0][1];
```

```
wk[1][0] = wk[1][0] + h*k[1][0];
wk[1][1] = wk[1][1] + h*k[1][1];
}

//integracao numerica
void integracao(double x0, double x2, double *f_int_x0, double
*f_int_x1, double *f_int_x2, double *resultado){
int i;
double x1, h;
h = (x2-x0)/2.0;
x1 = (x2 + x0)/2.0;
for(i=0; i<2; i++)
resultado[i] = h*(f_int_x0[i]+4.0*f_int_x1[i]+f_int_x2[i])/3.0;
}

//inversa da matriz
void inv_matriz(double **matriz, double **inversa){
double det;
det = determinante_2x2(matriz);
inversa[0][0] = matriz[1][1]/det;
inversa[0][1] = -matriz[0][1]/det;
inversa[1][0] = -matriz[1][0]/det;
inversa[1][1] = matriz[0][0]/det;
}

//determinante
double determinante_2x2(double **matriz){
```

```
return matriz[0][0] * matriz[1][1] - matriz[0][1] * matriz[1][0];
}

//multiplicacao de uma matriz 2x2 por uma 2x1 (vetor tam 2)
void mult_2x2(double **matriz, double *vetor, double *resultado){
resultado[0] = matriz[0][0]*vetor[0] + matriz[0][1]*vetor[1];
resultado[1] = matriz[1][0]*vetor[0] + matriz[1][1]*vetor[1];
}

//tranforma em um vetor nulo
void zera_vetor(double *vetor){
int i;
for(i=0; i<2; i++)
vetor[i] = 0.0;
}

//tranforma em uma matriz nula
void zera_matriz(double **matriz){
int i, j;
for(i=0; i<2; i++)
for(j=0; j<2; j++)
matriz[i][j] = 0.0;
}

//tranforma em uma matriz unitaria
void matriz_unitaria(double **matriz){
matriz[0][0] = 1; matriz[0][1] = 0; matriz[1][0] = 0; matriz[1][1]
```

```
= 1;
}
```

Referências

[1] AUERNIG, J.W.; TROGER, H. Time optimal control of overhead cranes with hoisting of the load. **Automatica**, v.23, n.4, p. 437-447, 1987.

[2] CHENG C.Y. The switching algorithm for the control of overhead crane. **Neural Computing & Applications**, v.15, n.3-4, p. 350-358, 2006.

[3] SAKAWA, Y.; SHINDO, Y. Optimal control of container cranes. **Automatica**, v.18, n.3, p. 257-266, 1982.

[4] AL-GARNI, A.Z.; MOUSTAFA, K.A.F.; JAVEED NIZAMI, S.S.A.K. Optimal control of overhead cranes. **Control Eng. Practice**, v.3, n.9, p. 1277-1284, 1995.

[5] FIGUEIREDO, D.G. **Equações diferenciais aplicadas**. 2.ed. Rio de Janeiro: IMPA, 2005. 307 p.

[6] OGATA, K. **Discrete-time control systems**. 2.ed. Englewood Cliffs, N.J. : Prentice Hall, 1995. 745 p.

[7] BURDEN, R.L.; FAIRES, J. D. **Análise numérica**. São Paulo : Pioneira Thomson Learning, 2003. 736 p.

[8] LUENBERGER, D. G. **Linear and nonlinear programming**. 2.ed. Boston : Kluwer Academic Publishers, 2003. 491 p.

[9] GRACE, A. et al. **Optimization toolbox for use with MATLAB**. Natick: MathWorks, 1992. 180 p.