Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

Ruy Dantas Nóbrega

Programa Didático de Simulação e Identificação de Sistemas

Campina Grande, Paraíba

Março de 2013

Ruy Dantas Nóbrega

Programa Didático de Simulação e Identificação de Sistemas

Trabalho de Conclusão de Curso submetido à
Unidade Acadêmica de Engenharia Elétrica
Universidade Federal de Campina Grande
como parte dos requisitos necessários para a
obtenção do grau de Bacharel em Ciências no
Domínio da Engenharia Elétrica

Área de concentração: Controle Analógico, Controle Digital e Identificação de Sistemas

Orientador:
Professor João Batista Morais dos Santos, Dr.
Campina Grande, Paraíba
Março de 2013

Ruy Dantas Nóbrega

Programa Didático de Simulação e Identificação de Sistemas

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica

Área de concentração: Controle Analógico, Controle Digital e Identificação de Sistemas

Aprovado em / /

Ruy Dantas Nóbrega
Universidade Federal de Campina Grande
Aluno

Professor João Batista Morais dos Santos, Dr.
Universidade Federal de Campina Grande
Orientador

Dedico este trabalho à todos que duvida-
ram deste dia.

Agradecimentos

Agradeço à toda minha família, em especial à minha mãe, pai e irmã, que sempre me apoiaram sobre os momentos mais difíceis passados durante o curso.

À minha namorada e melhor amiga, Dayanne Rocha, por estar sempre ao meu lado, nas horas boas e ruins.

À todos os meus amigos, os antigos, e os novos que encontrei pelo caminho: Victor Loudal, Mário Júnior, Rubem Danilo, Pablo Vinícius, Carlos Ângelo, Lucas Borges, Marconni Freitas, Felipe Lucena, Juliano Leal, João Marcelo e muitos outros.

Finalmente, agradeço à meu orientador, João Batista, por dar-me a oportunidade, com este projeto, de retribuir um pouco à universidade tudo que esta me ofereceu durante todo o curso.

”A ciência é uma perversão de si mesma, a menos que tenha como fim último, melhorar a humanidade” - Nikola Tesla

Resumo

Neste trabalho é descrito o projeto do Programa Didático de Simulação e Identificação de Sistemas (PDSIS). Este programa objetiva permitir a simulação e identificação de sistemas de maneira simples e rápida, sem a necessidade de conhecimentos de programação por parte do usuário. O projeto foi elaborado no ambiente MATLAB® , o que agilizou o processo de desenvolvimento do programa devido à linguagem simples e eficiente. Testes foram projetados de forma a utilizar todos os recursos disponíveis do programa e, então, executados para comprovar a estabilidade e a viabilidade deste para o uso em sala de aula.

Palavras-Chave: Identificação, simulação, experimento, sistema, função de transferência, programa, MATLAB

Abstract

This paper describes the Didactical Systems Simulation and Identification Software (PDSIS). This program have the systems' simulation and identification simply and quickly as the main objective, requiring no programming knowledge from the user. The project was designed in the MATLAB® environment, which sped up the program development process due to the simple and efficient language. Tests were designed to use all available resources of the program and then executed to verify the stability and viability of this for use in the classroom.

Key words: Identification, simulation, experiment, system, transfer function, software, MATLAB

Sumário

1 Introdução

Em teoria, é possível modelar qualquer sistema dinâmico a partir de seus elementos e suas equações características e projetar um controlador apropriado para uma resposta desejada. Mas, quando colocado na prática, o sistema modelado muitas vezes não responde como esperado devido à várias razões, tais como ruídos e elementos não-perfeitos, diferindo da modelagem fenomenológica obtida inicialmente. E para sistemas mais complexos (ou totalmente desconhecidos, como o modelo caixa preta), é ainda mais árduo e menos eficiente o trabalho de modelagem, tornando-se necessário o uso de outros métodos para a obtenção dos modelos. Deste modo, a identificação de sistemas dinâmicos por dados experimentais se torna essencial para o controle de processos.

Há diversos programas disponíveis para a realização de identificação de sistemas, como o ITSIE desenvolvido por J. L. Guzmán (2012), mas nenhum didático e simples o suficiente para o uso em aulas práticas e rápidas demonstrações. O uso destes programas, devido às suas complexidades, limitações e especificidades, em aula desperdiçaria tempo e interesse dos alunos, que seriam melhores aproveitados explorando o estudo de identificação de sistemas e sua relação com a teoria. Por isso, a elaboração de um programa projetado especificamente para o uso didático se faz necessário.

O programa aqui proposto objetiva aproximar a teoria à prática de simulação e identificação de sistemas, eliminando o custoso trabalho de cálculo e programação que liga estas duas. Através de uma interface simples e de fácil assimilação, o Programa Didático de Simulação e Identificação de Sistemas (PDSIS) procura eliminar, para o aluno, a necessidade de conhecimentos prévios em programação MATLAB® .

2 Fundamentos Teóricos

2.1 Identificação de Sistemas

A identificação de sistemas surgiu da necessidade de modelar matematicamente um sistema desconhecido baseado apenas na análise de dados obtidos a partir de medições das entradas e saídas. Seus princípios são importados da área de estatística, como o método dos mínimos quadrados e máxima verossimilhança.

Nesta seção será apresentado uma introdução sobre a identificação de sistemas, introduzindo o modelo ARX e a generalização deste modelo para o modelo Box-Jenkins (BJ).

2.1.1 Modelo ARX

Inicialmente, pode-se representar um sistema pela a equação diferencial discreta

$$y(t)+a_1y(t-\Delta t)+...+a_ny(t-n\Delta t) = b_1u(t-\Delta t)+b_2u(t-2\Delta t)+...+b_mu(t-m\Delta t) \quad (2.1)$$

em que $u(t)$ representa a entrada, $y(t)$ e t o tempo. Δt é o período de amostragem. A fins de simplificação, é considerado $\Delta t = 1$ e $t \in Z^+$. Como é visto, os valores de $y(t)$ são dependentes tanto dos valores das entradas passadas como dos valores passados da saída. Por isto, este modelo, que possui autoregressão parcial, tem o nome de ARX (*Auto-Regression with eXtra inputs*), do inglês, autoregressão com entradas extras.

Outra forma de se ver a Equação (2.1), é determinar a saída baseado nos valores passados das entradas e saídas:

$$y(t) = -a_1y(t-1) - ... - a_ny(t-n) + b_1u(t-1) + b_2u(t-2) + ... + b_mu(t-m). \quad (2.2)$$

Deste modo, $y(t)$ pode ser representado pela multiplicação de dois vetores:

$$y(t|\theta) = \varphi(t)\theta \quad (2.3)$$

em que:

$$\varphi(t) = \begin{bmatrix} -y(t-1) & ... & -y(t-n) & u(t-1) & ...u(t-m) \end{bmatrix} \quad (2.4)$$

$$\theta = \begin{bmatrix} a_1 & ... & a_n & b_1 & ... & b_m \end{bmatrix}^T \tag{2.5}$$

$y(t|\theta)$ indica a dependência de y à t, dado um determinado θ.

2.1.1.1 Método dos minímos quadrados

Considerando-se agora que um sistema, em que não é sabido os valores de θ, foi experimentado e teve seus dados de saída e entrada gravados durante N tempos de amostragem, formando o conjunto

$$Z^N = \{u(1), y(1), u(2), y(2), ..., u(N), y(N)\}. \tag{2.6}$$

Assim, um dos critérios para se determinar os valores do vetor θ é a minimização de uma função custo do tipo:

$$V(\theta, Z^N) = \frac{1}{N}\sum_{t=1}^{N}(y(t) - \varphi(t)\theta)^2. \tag{2.7}$$

em que $V(\theta, Z^N)$ significa a média dos quadrados dos erros entre os valores obtidos experimentalmente e um sistema modelado para um dado θ. Assim, para encontrar o θ que minimiza $V(\theta, Z^N)$ utiliza-se o Teorema do Valor Mínimo, derivando $V(\theta, Z^N)$ em função de t e igualando-o a zero:

$$\frac{dV(\theta, Z^N)}{dt} = \frac{2}{N}\sum_{t=1}^{N}\varphi(t)^T(y(t) - \varphi(t)\theta) = 0 \tag{2.8}$$

isolando θ,

$$\theta = \big[\sum_{t=1}^{N}\varphi(t)^T\varphi(t)\big]^{-1}\sum_{t=1}^{N}\varphi(t)^T y(t). \tag{2.9}$$

Assim, a partir da Equação (2.9) pode-se calcular o valor de θ que minimiza $V(\theta, Z^N)$.

2.1.2 Modelo Box-Jenkins

O modelo Box-Jenkins tem o próposito de generalizar a equação diferencial que representa um sistema. Considere um sistema linear que receba uma entrada $u(t)$ mais uma pertubação $e(t)$:

$$y(t) = G(q, \theta)u(t) + H(q, \theta)e(t) \tag{2.10}$$

em que $G(q, \theta)$ é a função de transferência da entrada para a saída escrita em função de q, o operador deslocamento. A pertubação $e(t)$ é assumida que passa por um "filtro" com função de transferência $H(q, \theta)$. Como funções de transferências, estas são esperadas para serem divisões polinomiais e , assim, podem ser reescritas:

$$G(q, \theta) = \frac{B(q)}{F(q)} \tag{2.11}$$

$$H(q, \theta) = \frac{C(q)}{D(q)} \tag{2.12}$$

Então $y(t)$ pode ser escrito na forma

$$y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t) \tag{2.13}$$

A Equação (2.13) é o modelo "geral" de representação de sistemas e é conhecida como o modelo de Box-Jenkins, proposto por George E. P. Box e GWilym M. Jenkins.

Este modelo é considerado como o modelo geral porque outros métodos podem ser considerados como casos particulares.

2.1.3 Modelo OE

O modelo OE (*Output Error*), por exemplo, é um caso particular de BJ quando $C(q) = 1$ e $D(q) = 1$, resultando no modelo

$$y(t) = \frac{B(q)}{F(q)}u(t) + e(t) \tag{2.14}$$

2.1.4 Modelo ARMAX

O modelo ARMAX (*Average Model with eXternal inputs*), é um caso particular de BJ quando $F(q) = D(q) = A(q)$, resultando no modelo

$$A(q)y(t) = B(q)u(t) + C(q)e(t) \tag{2.15}$$

O modelo ARX já é um caso particular do modelo ARMAX, quando $C(q) = 1$:

$$A(q)y(t) = B(q)u(t) + e(t) \tag{2.16}$$

2.1.5 Modelo FIR

O modelo FIR (*Finite Impulse Response*) é um caso particular de ARX quando $A(q) = 1$ resultando na equação

$$y(t) = B(q)u(t) + e(t) \tag{2.17}$$

2.2 Programação em MATLAB®

MATLAB® é um programa de propriedade da empresa MathWorks® voltado para a resolução de problemas numéricos das mais diversas áreas como aeronática, circuitos elétricos, estatística, controle analógico e controle digital.

Nesta seção será apresentado uma introdução à simulação de sistemas analógicos e digitais, identificação de sistemas e interfaces gráficas em MATLAB® .

2.2.1 Simulação de sistemas

A resolução de sistemas analógicos e digitais em MATLAB® é simples e direta. Sabendo-se o vetor do sinal de entrada e do tempo, pode-se simular sistemas analógicos utilizando apenas o comando `lsim()`. O código a seguir ilustra o uso deste comando para obter a resposta de um sistema analógico à uma entrada senoidal:

```
t = [0:0.001:5];
u = sin(2*pi*t);
planta = tf(1,[1 1]);
y = lsim(planta, u, t);
```

Para simulações de sistemas digitais, o processo é similar, utilizando o comando `dlsim()`. O código a seguir ilustra o uso deste comando para obter a resposta de um sistema digital

```
t = [0:0.001:5];
u = sin(2*pi*t);
```

```
planta = tf(9.99e-4,[1 -0.999], 0.001);
y = dlsim(planta, u, t);
```

2.2.2 Identificação de sistemas

Para a identificação de sistemas, MATLAB® proporciona métodos nativos de identificação para os modelos mais utilizados. São eles:

`arx(data, orders)` Utiliza os dados de `data` para obter um modelo ARX. `order` é um vetor com as ordens dos polinômios $A(q)$ e $B(q)$ e o atraso na entrada K em número de amostragens, respectivamente.

`bj(data, orders)` Utiliza os dados de `data` para obter um modelo BJ. `order` é um vetor com as ordens dos polinômios $B(q)$, $C(q)$, $D(q)$ e $F(q)$ e o atraso na entrada K em número de amostragens, respectivamente.

`oe(data, orders)` Utiliza os dados de `data` para obter um modelo OE. `order` é um vetor com as ordens dos polinômios $B(q)$ e $F(q)$ e o atraso na entrada K em número de amostragens, respectivamente.

`armax(data, orders)` Utiliza os dados de `data` para obter um modelo ARMAX. `order` é um vetor com as ordens dos polinômios $A(q)$, $B(q)$ e $C(q)$ e o atraso na entrada K em número de amostragens, respectivamente.

Para a identificação utilizando o modelo FIR, utiliza-se o comando `arx()`, com a ordem de $A(q)$ igual a um. Em todos os comandos, `data` é um objeto do tipo `iddata`, que guarda os valores do sinal de entrada, do sinal de saída e o período de amostragem. O código a seguir identifica um sistema utiilizando os cinco modelos descritos com dados carregados a partir de arquivos.

```
u = load('c:/entrada.txt','-ascii');
y = load('c:/saida.txt', '-ascii');
dados = iddata(y, u, 0.001);
mbj = bj(dados, [2 3 4 5 6]);
moe = oe(dados, [2 3 5]);
marmax = armax(dados, [2 3 4 5]);
marx = arx(dados, [3 4 5]);
```

```
mfir = arx(dados, [1 4 5]);
```

2.2.3 Interfaces Gráficas

Além de resolução rápida de problemas numéricos, MATLAB® permite que sejam programadas GUIs (*Graphic User Interface*), que são janelas que podem conter botões, campos de texto, listas, *checkboxes*, gráficos entre outros.

As GUIs em MATLAB® são programadas utilizando-se o objeto `figure`, que conterá os outros objetos. Pode-se dizer que `plot()` é um caso particular de `figure`, onde este contém apenas um gráfico.

GUIs são tratadas como funções, que podem receber parâmetros iniciais. Para o tratamento de eventos, o MATLAB® usa o conceito de *callbacks*, que são funções dentro da função principal, que são chamadas quando uma ação ocorre. O código a seguir cria uma GUI com um botão, uma caixa de edição de texto e um objeto de texto. O botão tem a simples função de copiar o que há na caixa de texto para o objeto de texto.

```
function GUIExemplo
    GUI = figure('MenuBar', 'none',...
            'ToolBar', 'none',...
            'Position', [0,0,270,100],...
            'Name', 'Teste',...
            'NumberTitle', 'off',...
            'Resize', 'off');
    Texto = uicontrol( 'Style', 'text',...
            'String', ' ',...
            'TAG', 'TXT',...
            'Position', [10,80,250,20]);
    Edit = uicontrol( 'Style', 'edit',...
            'String', ' ',...
            'TAG', 'EDIT',...
            'BackgroundColor', [1 1 1],...
            'Position', [10,55,250,20]);
    Botao = uicontrol( 'Style', 'pushbutton',...
```

```
            'String', 'Copiar',...
            'Position', [10,10,250,40],...
            'Callback', @BtnSimClicked);
    function BtnSimClicked(Objeto, Evento)
        set(findall(GUI, 'TAG', 'TXT'), 'String',...
        get(findall(GUI, 'TAG', 'EDIT'),'String'));
    end
end
```

NA Figura 2.1 é mostrada a janela gerada por este código.

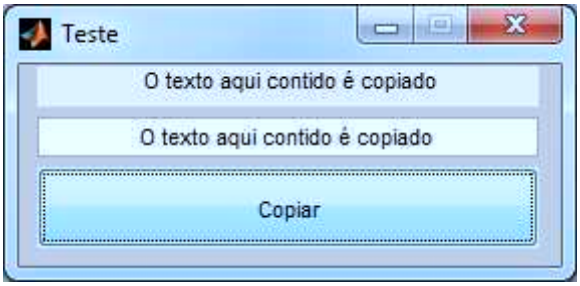


Figura 2.1 – Figura gerada pela função `GUIExemplo`

Este código representa de maneira satisfatória como um código GUI é gerado em MATLAB® . No código, `GUI` é um objeto do tipo `figure` que tem várias de suas propriedades modificadas para se adaptar ao que é requerido como: não aparecer o menu padrão do MATLAB® , não mostrar a caixa de ferramentas, mudar a posição e tamanho da janela, mudar o título da janela e não mostrar o número da figura.

De modo similar, o objeto de texto, a caixa de edição de texto e o botão são criados a partir do comando `uicontrol` que também permiter modificar vários aspectos dos objetos como, por exemplo, a cor de fundo do objeto de edição de texto e qual função será chamada quando o botão for pressionado. Nota-se que os comandos `uicontrol` não recebem `GUI` como parâmetro. O MATLAB® assume que os objetos criados por esta função serão colocados na figura que "está em foco", neste caso, `GUI`.

A função *callback* do botão mostra os comandos `set()`, `get()` e `findall()` sendo utilizados para copiar o texto da caixa de edição de texto para o objeto de texto.

O `findall()` permite procurar por objetos dentro de uma figura a partir de suas propriedades. No exemplo, o comando está sendo primeiramente usado para procurar os objetos que tenham a propriedade `TAG` igual a `TXT`, que é apenas o objeto de texto.

O comando `get()` é utilizado para obter o valor de determinada propriedade de um objeto. No exemplo, este comando está sendo utilizado em conjunto com `findall()` para obter o texto (*string*) contido na caixa de edição de texto.

O comando `set()` é utilizado para modificar uma ou mais propriedades de um objeto. O primeiro parâmetro é sempre o controle (*handle*) do objeto. Os demais parâmetros são os nomes das propriedades que se deseja modificar e o novo valor para elas. No exemplo, é procurado o objeto de texto e seu texto (*string*) recebe o texto que está escrito na caixa de edição de texto.

3 Descrição do Programa

Para a programação do Programa Didático de Simulação e Identificação de Sistemas (PDSIS), foi utizada a linguagem MATLAB® . Esta foi escolhida pela sua flexibilidade na manipulação de matrizes e rápido desenvolvimento. Em comparação a outras linguagens de programação possíveis, como C/C++, Pascal, FORTRAN e Basic, MATLAB® oferece uma excelente plataforma para cálculos numéricos e, construído sobre a plataforma *Java™ Runtime Environment* (JRE), MATLAB® liberta o programador do gerenciamento de memória, que é feito pelo *Garbage Collector*(*gc*) de Java™ , acelerando o desenvolvimento do programa.

3.1 Definição de Classes

À primeira vista, há dois tipos de classes: `simulação` e `experimento`. A classe `simulação` guarda informações sobre simulações, como a equação do sinal de entrada, a planta do sistema, o período de amostragem e os tempos iniciais e finais da simulação. A classe `experimento` contém resultados experimentais em forma de vetores de entrada e saída, e o período de amostragem.

Apesar da diferente natureza entre as duas classes, elas possuem diversos parâmetros e métodos em comum. Assim, decidiu-se por fundí-las em apenas uma, a classe `sistema`. Nesta classe é guardada todas as informações necessárias sobre qualquer simulação ou experimento, sendo a diferenciação entres elas feita internamente. O código de definição da classe é mostrado a seguir.

```
classdef sistema
    properties
        caminho = ' ';
        arquivo = ' ';
        modificado = '*';
        visualizar = '<O>';
        estilo = 'b';
        versao=1;
        nome = ' ';
        tini = 0;
        tpas = 0.1;
```

```
        tfin = 1;
        digital = 0;
        entrada = NaN;
        planta = NaN;
        entradapos = [];
        plantapos = [];
        resposta = [];
    end
    methods
        function obj = sistema(varargin);
        function disp(obj);
        function obj = resolver(obj);
        function obj = salvar(obj, pathname_, filename_);
        function obj = carregar(obj, filename_);
    end
end
```

As funções e variáveis desta classe são definidas como:

caminho Diretório onde o arquivo contendo as informações do sistema é salvo.
arquivo Nome do arquivo contendo as informações do sistema.
modificado Informação se o atual sistema difere do sistema salvo em arquivo, ou se não foi ainda salvo. O valor `'*'` indica que o sistema foi modificado, `' '` caso contrário.
visualizar Informação se o sistema é mostrado ou não na interface gráfica. O valor `'<O>'` indica que o sistema é visível, `' '` caso contrário.
estilo Cor do desenho do gráfico. pode ser preto (k), azul (b), vermelho (r), verde (g), ciano (c), magenta (m) ou amarelo (y).
versao Versão de codificação do arquivo onde está/será salvo as informações do sistema. Esta variável será utilizada para posteriores atualizações.
nome Nome do sistema.
tini Tempo inicial da simulação/experimento. Quando experimento, é sempre zero.
tpas período de amostragem da simulação/experimento.
tfin Tempo final da simulação/experimento. Quando experimento, é definido pelo período de amostragem.
digital Indica se o sistema é digital ou analógico. 0 se analógico, 1 se digital.

entrada Equação do sinal de entrada em forma de texto.

planta Equação da planta do sistema em forma de texto. Quando experimento, seu valor é `NaN` (acrônimo inglês para *Not a Number*).

entradapos Vetor que guarda os valores processados da equação do sinal de entrada em forma de texto.

plantapos Objeto do tipo `tf`, que guarda a função de transferência da planta a partir da equação da planta em forma de texto.

resposta Vetor que guarda os valores resultantes da simulação/experimento.

sistema(varargin) Construtor da classe. Recebe seis parâmetros se for uma simulação: nome, entrada, planta, tempo inicial, período de amostragem e tempo final. E quatro parâmetros caso seja um experimento: nome, vetor do sinal de entrada, vetor do sinal de saída e o período de amostragem. Pode também receber apenas um parâmetro, para criar o objeto a partir de um arquivo.

disp(obj) Sobrecarga do método `disp()` para a classe `sistema`.

resolver(obj) Função para executar e guardar os resultados de simulação. Não funciona para experimentos.

salvar(obj, pathname_, filename_) Salva o sistema no arquivo `filename_` no diretório `pathname_`.

carregar(obj, filename_) Carrega os dados guardados no arquivo `filename_`.

A versatilidade desta classe permite que ela seja utilizada tanto para sistemas analógicos como digitais, e tanto para simulação como para resultados experimentais. Pode-se interpretar a relação simulação-experimento como ‘experimento’ sendo uma subclasse de ‘simulação’, pois é necessário apenas parte dos métodos e das variáveis. O diagrama da Figura 3.1 ilustra esta relação.

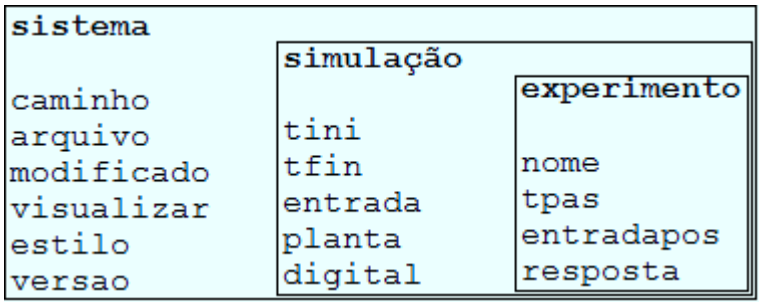


Figura 3.1 – Diagrama de relação de classes simulação-experimento

Internamente, a classe `sistema` diferencia simulações de experimentos através da variável `planta`. Quando `planta` é uma cadeia de caracteres (do tipo `string`),

o objeto é uma simulação mas, caso o valor seja `NaN`, o objeto representa um experimento.

A variável `planta` também permite diferenciar um sistema analógico de um digital. Esta define o valor da variável `digital`, que é uma variável suplementar à `planta`. Se a equação da planta for escrita em função de 's', o objeto representa a simulação de um sistema analógico e, por isso, `digital` recebe o valor zero e simula-se o sistema utilizando o comando `lsim`. Mas, caso a planta esteja escrita em função de 'z', isto indica que o objeto representa a simulação de um sistema digital, então `digital` recebe o valor hum e se é simulado utilizando o comando `dlsim`. Quando se é um experimento, assume-se o valor hum para a variável `digital`.

3.1.1 Codificação do Arquivo .pdsis

As informações instanciadas da classe `sistema` podem ser salvas em (ou carregadas de) arquivos definidos com extensão `.pdsis`, utilizando o método `salvar()` (ou `carregar()`). Nestes arquivos, estão contidas todas as informações fundamentais sobre o objeto em forma de texto.

A codificação deste tipo de arquivo é bastante simples. Todos os dados dentro do arquivo são separados por `;` (ponto e vírgula). Os dados são arranjados de acordo com o Quadro 3.1.

Simulações e experimentos são guardados de maneiras diferentes no arquivo. Para se instanciar um objeto de `sistema` como simulação, é necessário que este objeto possua a equação da planta e do sinal de entrada, para que o sistema possa ser simulado. Assim, é guardado, no arquivo, a equação da planta da simulação na posição 8 e, na posição 9, é guardado a equação do sinal de entrada.

Já experimentos são guardados de forma diferente. Experimentos são apenas vetores de dados da entrada e da resposta do sistema e, por isso, não possuem planta. Assim, no arquivo, a posição número 8 sempre será 'NaN', e as posições de 9 a 2n+8 guardam os valores númericos dos vetores da entrada e da resposta.

3.2 O *Parser* do PDSIS

A principal característica pensada para o desenvolvimento do PDSIS foi manter a simplicidade para tornar a experiência de uso a mais fácil possível. Assim, um *parser* foi desenvolvido exclusivamente para o uso no PDSIS.

Quadro 3.1 – Arranjo de dados nos arquivos `.pdsis`

Posição	**Dado**	Descrição
1	`versão`	Versão de codificação do arquivo.
2	`nome`	Nome de `sistema`.
3	`tini`	Tempo incial.
4	`tpas`	Tempo do amostragem.
5	`tfin`	Tempo final.
6	`estilo`	Estilo de gráfico.
7	`digital`	Indicador se `sistema` é ou não digital.
8	`planta`	Planta de `sistema`: Texto se for simulação, NaN (*Not a Number*) se for experimento.
9	`entrada`	Entrada em forma de texto. Existe apenas se for simulação.
9..n+8	`entradapos`	Vetor numérico de tamanho n para entrada. Existe apenas se for experimento.
n+9..2n+8	`resposta`	Vetor numérico de tamanho n para resposta. Existe apenas se for experimento.

Parser, em computação, é um analisador de sintaxe que transforma uma entrada, geralmente textual, em uma estrutura de dados úteis ao programa. O MATLAB® possui um excelente *parser* para interpretar as equações e expressões que são escritas em seu console, ou arquivos-m, e resolvê-las, retornando matrizes como resultado.

O *parser* do PDSIS é utilizado para interpretar a equação em forma de texto do sinal de entrada quando a instância de `sistema` é uma simulação. A motivação de programar um *parser* próprio à utilizar o do MATLAB® , foi permitir escrever equações mais próximas à forma matemática possível e incluir funções que não são definidas diretamente em MATLAB® .

A exemplo, considere o código abaixo, com `t = [1:10]`:

```
eq1 = 't.^2';
eq2 = 't./(t+1)';
eq3 = 'idinput(length(t), "prbs", [0 0.5])';
```

Em MATLAB® , pode-se resolver estas equações em forma de texto utilizando apenas o método `eval()`. Mas, apesar de simples para usuários medianos de MATLAB® , é desinteressante para o PDSIS, o usuário necessitar saber o significado do ponto após a variável `t` em `eq1` e `eq2`, ou o que todos os parâmetros de `idinput()` signi-

ficam em `eq3`. Assim, um método mais fácil de se escrever estas mesmas equações seria aproximá-las à sua forma matemática:

```
eq4 = 't^2';
eq5 = 't/(t+1)';
eq6 = 'prbs(0.5)';
```

Mas, as equações escritas nesta forma não podem ser processadas pelo *parser* do MATLAB® , pois não há o operador elemento-a-elemento (o ponto) em `eq4` e `eq5`, e o método `prbs()` em `eq6` não é definido em MATLAB® . Deste modo, foi construído o *parser* do PDSIS para interpretar este tipo de escrita, o `str2eq`.

Internamente, o *parser* `str2eq` utiliza a função `eval()` para resolver as equações, substituindo a variável `t` por seu valor correspondente e pré-processando as funções não definidas em MATLAB® . As funções aceitas pelo *parser* são:

`deg(t)` Função degrau. Quando `t` for maior ou igual a zero, a função retorna 1, caso contrário, 0.

`imp(t)` Função impulso de Dirac. Quando `t` for maior ou igual a zero, a função retorna um único pulso de amplitude $1/\Delta t$, onde Δt é o período de amostragem.

`prbs(w)` Gera um sinal prbs, onde `w` é a frequência máxima, entre 0 e 1.

`triang(t,q)` Gera uma onda triangular, onde `t` é o tempo e `q` é a relação de trabalho da onda. Quando `q=1` é gerado uma onda dente-de-serra. Quando `q=0` é gerado uma onda dente-de-serra invertida. Quando `q=0.5` é gerado uma onda triangular simétrica.

`abs(t)` Função módulo. Retorna o valor abstoluto de `t`.

`sin(t)` Função seno.

`cos(t)` Função cosseno.

`tan(t)` Função tangente.

`asin(t)` Função arco-seno.

`acos(t)` Função arco-cosseno.

`atan(t)` Função arco-tangente.

`sqrt(t)` Função raiz quadrada.

`exp(t)` Função exponencial.

`log(t)` Função logarítmica com base neperiana.

`log2(t)` Função logarítmica com base 2.

`log10(t)` Função logarítmica com base 10.

A função de onda quadrada não é diretamente implementada, mas pode ser obtida com o seguinte código:

```
eq7 = 'deg(triang(2*pi*t,0))';
```

Deste modo, gera-se uma onda quadrada com período de hum segundo. Vale notar que também pode-se gerar um sinal PWM (acrônimo inglês para *Pulse Width Modulation*) somando um sinal constante, entre -1 e 1 à função `triang()`.

A função chirp, que é um seno de frequência variável, também não é definida mas pode ser obtida da seguinte forma:

```
eq8 = 'sin(2*pi*(df*t^2+fi*t));'
```

Onde `fi` é a frequência inicial e `df` é a variação de frequência por segundo. Assim, caso queira-se um seno que varia de 20Hz a 100Hz em 2 segundos, `fi=20` e `df=80/2=40`.

3.3 Instalação do PDSIS

A instalação do PDSIS é simples e rápida. O arquivo de instalação é de extensão `.rar` e contém apenas uma pasta, `PDSIS`, como mostrado na Figura 3.2.

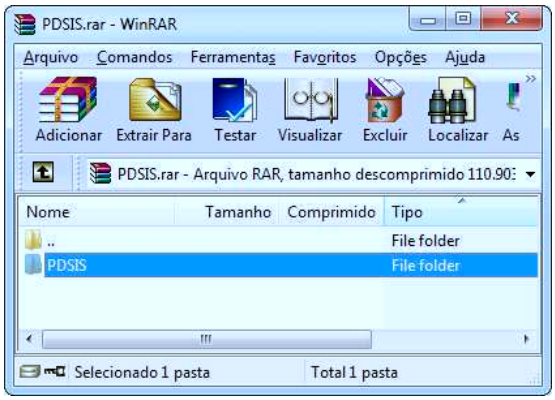


Figura 3.2 – Arquivo `PDSIS.rar` aberto com o programa *Winrar*, contendo a pasta `PDSIS`

Como administrador da máquina, a pasta `PDSIS` deve ser descompactada dentro da pasta `<MATLAB>/toolbox`. onde `<MATLAB>` é o caminho da pasta onde está instalado o MATLAB® , como por exemplo `c:/Arquivos de Programas/MATLAB/`

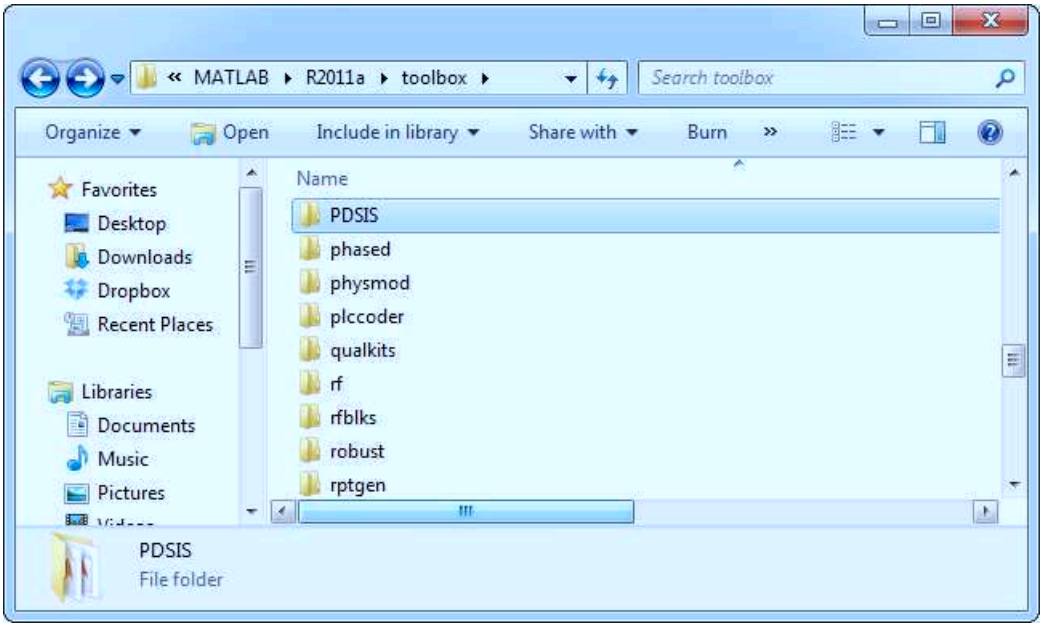


Figura 3.3 – Pasta `PDSIS` descompactada dentro da pasta `toolbox`

`R2011a/`. A Figura 3.3 mostra a pasta `PDSIS` já descompactada dentro da pasta `toolbox`.

A seguir, executa-se o MATLAB® como administrador, clicando com o botão direito no ícone do MATLAB® e então em *Executar como Administrador*. Então, para o último passo da instalação, digita-se no console os comandos:

```
» addpath('<MATLAB>/toolbox/PDSIS')
» savepath
```

em que `<MATLAB>` deve ser trocado pelo caminho onde o MATLAB® está instalado. Estes comandos adicionam a pasta do PDSIS ao `path` do MATLAB® e salvam esta modificação. A execução como administrador garante a execução bem sucedida destes comandos.

Após estes passos, o PDSIS está completamente instalado no MATLAB® e pode ser executado a partir do console digitando o comando:

```
» PDSIS
```

3.4 Interface Gráfica

Após instalado e iniciado sua execução, o PDSIS abre a janela principal, composta apenas de gráficos, uma lista e duas barras de ferramentas. A Figura 3.4 mostra a janela principal do programa com as partes principais enumeradas.

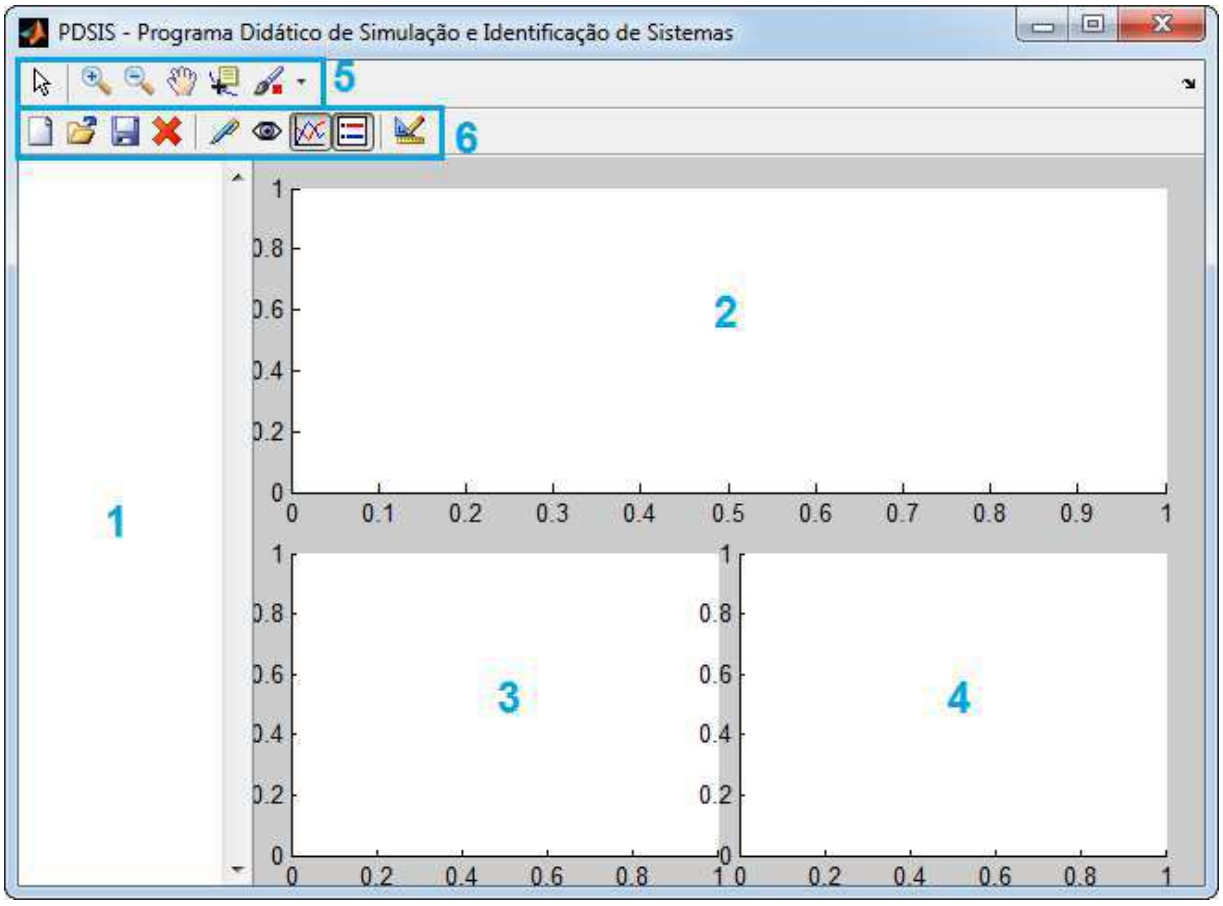


Figura 3.4 – Janela principal do PDSIS

A área 1 é a lista de sistemas (leia-se 'sistemas', como instâncias da classe `sistema`). Nesta lista são colocados todos os sistemas que estão sendo utilizados no PDSIS.

As áreas 2, 3 e 4 são os gráficos de resposta do sistema ao sinal de entrada no tempo, de diagrama de Bode, e de diagrama de Nyquist, respectivamente. Estes gráficos são baseados nos sistemas da lista de sistemas (área 1).

A área 5 é a caixa de ferramentas para manipulação dos gráficos. Esta caixa de ferramentas é padrão do MATLAB® e foi incorporada ao PDSIS. O Quadro 3.2 descreve

todos os botões e suas respectivas funções.

Quadro 3.2 – Descrição da barra de ferramentas do MATLAB®

Ícone	Comando	Descrição
	Editar Gráfico	Edita a área dos gráficos, como os limites do gráfico e cores.
	Ampliar Gráfico	Amplia uma área do gráfico.
	Reduzir Gráfico	Reduz a área ampliada.
	Manipular Gráfico	Manipula o gráfico ampliado
	Detalhar Dado	Detalha um dado selecionado de um gráfico
	Selecionar Dados	Seleciona um conjunto de dados

A área 6 é a caixa de ferramentas para manipulação de sistemas. Esta caixa de ferramentas é exclusiva e contêm todas as ações possíveis com o PDSIS. O Quadro 3.3 descreve todos os botões e suas respectivas funções.

Quadro 3.3 – Descrição da barra de ferramentas do PDSIS

Ícone	Comando	Descrição
	Novo Sistema	Cria um novo sistema.
	Abrir Sistema	Carrega um sistema a partir de um arquivo.
	Salvar Sistema	Salva um sistema em um arquivo.
	Excluir Sistema	Exclui o sistema selecionado da lista de sistemas.
	Editar Sistema	Edita o sistema selecionado da lista de sistemas.
	Visualizar Sistema	Mostra/esconde nos gráficos o sistema selecionado.
	Visualizar Entradas	Mostra/esconde as entradas de todos os sistemas.
	Visualizar Legendas	Mostra/esconde a legenda de todos os sistemas.
	Identificar Sistema	Identifica o sistema selecionado.

3.4.1 Interface de Criação e Edição de Sistemas

Para a criação de novos sistemas, deve-se clicar no botão *Novo Sistema*. Esta ação abre uma janela gráfica onde o usuário pode selecionar se deseja criar um sistema como simulação ou experimento. A Figura 3.5 mostra esta janela gráfica.

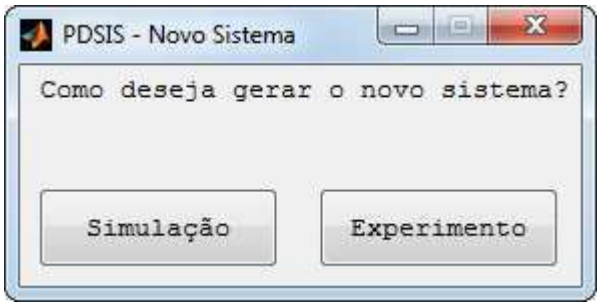


Figura 3.5 – Janela de seleção de novo sistema

Caso seja escolhido *Simulação*, uma nova janela é aberta, onde o usuário pode definir os parâmetros da simulação. A Figura 3.6 mostra a janela para a criação de uma simulação.

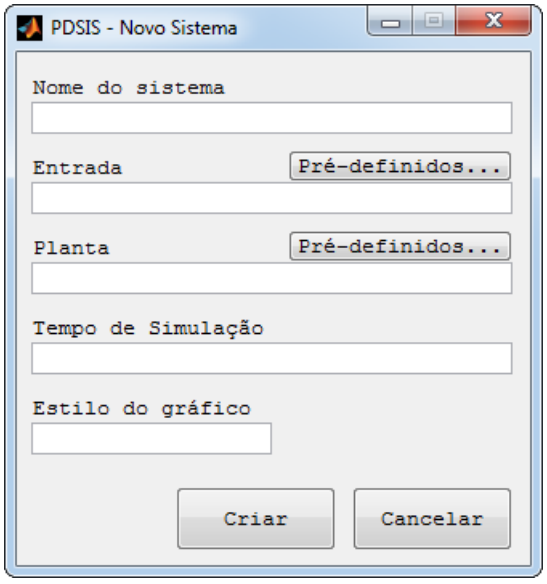


Figura 3.6 – Janela para criação de nova simulação

Os parâmetros necessários para criar uma nova simulação são definidos como:

`Nome` Nome da simulação para identificação.

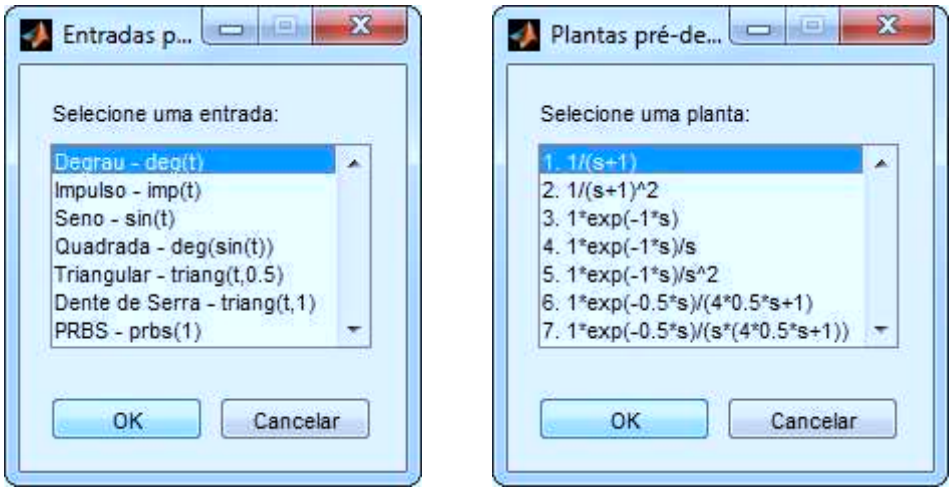
`Entrada` Equação do sinal de entrada em forma de texto. O método de escrita é o aceito pelo *parser* do PDSIS, em função de `t`.

`Planta` A planta para a qual o sistema será simulado em forma de texto, em função de `s` para analógico, e `z` para digital.

`Tempo de Simulação` O tempo pelo qual a simulação será executada. Deve ser no formato `<tempo inicial>:<período de amostragem>:<tempofinal>`. Por exemplo: `0:0.1:10`, simulará no tempo entre 0 e 10 segundos, com período de amostragem de 0,1 segundo.

`Estilo do Gráfico` Definição do estilo que é desenhado o gráfico, os valores aceitos são `c`, `y`, `m`, `k`, `r`, `g` ou `b`.

Há também dois botões na janela da Figura 3.6 entitulados *Pré-definidos...*, um na parte direita acima da caixa de edição de *Entrada* e o outro acima da caixa de edição de *Planta*. Estes botões permitem a seleção de entradas e plantas pré-definidas pelo PDSIS, a fim de acelerar e auxiliar a criação de novas simulações. Assim, quando o usuário escolhe uma função pré-definida, esta função é diretamente escrita no campo *Entrada* ou *Planta*. As janelas de seleção de funções pré-definidas para entradas e plantas são mostradas nas Figuras 7(a) e 7(b).



(a) Janela de seleção de entradas pré-definidas

(b) Janela de seleção de plantas pré-definidas

Figura 3.7

Para criar instâncias de `sistema` a partir de dados experimentais, clica-se no botão *Novo Sistema* e, na janela de seleção de sistema (Figura 3.5), clica-se em *Experimento*. Após isto, uma nova janela será mostrada para criação de sistemas via dados

experimentais. A janela para criação de sistemas via dados experimentais é mostrada na Figura 3.8.

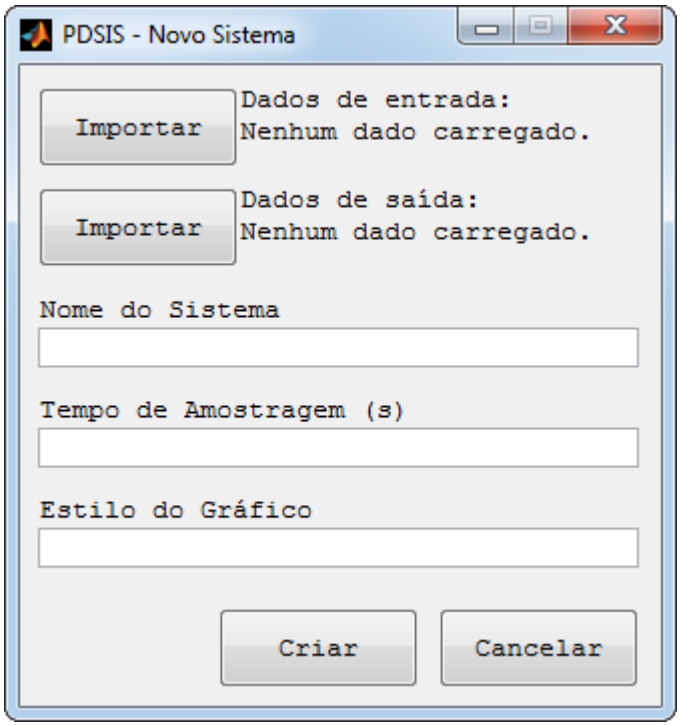


Figura 3.8 – Janela para criação de sistema via dados experimentais

Os parâmetros necessários para criar um novo sistema a partir de dados experimentais são, em parte, similares aos necessários para criar uma nova simulação. Os campos *Nome* e *Estilo de Gráfico* são campos idênticos aos campos para criação de uma nova simulação. O campo *período de amostragem* recebe apenas o período de amostragem do experimento, e o tempo final do experimento é calculado a partir deste, considerando o tempo inicial igual a zero. Os botões *Importar* executam a função de carregar, a partir da janela padrão do sistema operacional para seleção de arquivo, os arquivos contendo os dados experimentais de entrada (botão mais acima) e saída (botão abaixo).

Os dados experimentais no arquivo devem ser separados por uma linha apenas. Por exemplo, considere um vetor de 0 a 1, com o passo de 0,1. No arquivo, este deve estar escrito na forma:

```
0.0000
0.1000
```

```
...
0.9000
1
```

O PDSIS dá suporte a edição de sistemas depois que eles foram criados (ou carregados). Para tal, seleciona-se o sistema que se deseja editar na lista de sistemas (área 1 da janela da Figura 3.4) e, então, clica-se em *Editar Sistema*.

Dependendo do tipo de sistema (simulação ou experimento) que for selecionado para ser editado, o PDSIS abre a janela da Figura 3.6 ou da Figura 3.8, com todos os campos previamente preenchidos com os dados do sistema selecionado.

3.4.2 Interface de Identificação de Sistemas

Para acessar a janela de identificação de sistemas do PDSIS, deve-se primeiramente selecionar um sistema da lista de sistemas disponíveis e, então, clicar no botão *Identificar Sistema*. Na Figura 3.9 é mostrada a janela de identificação.

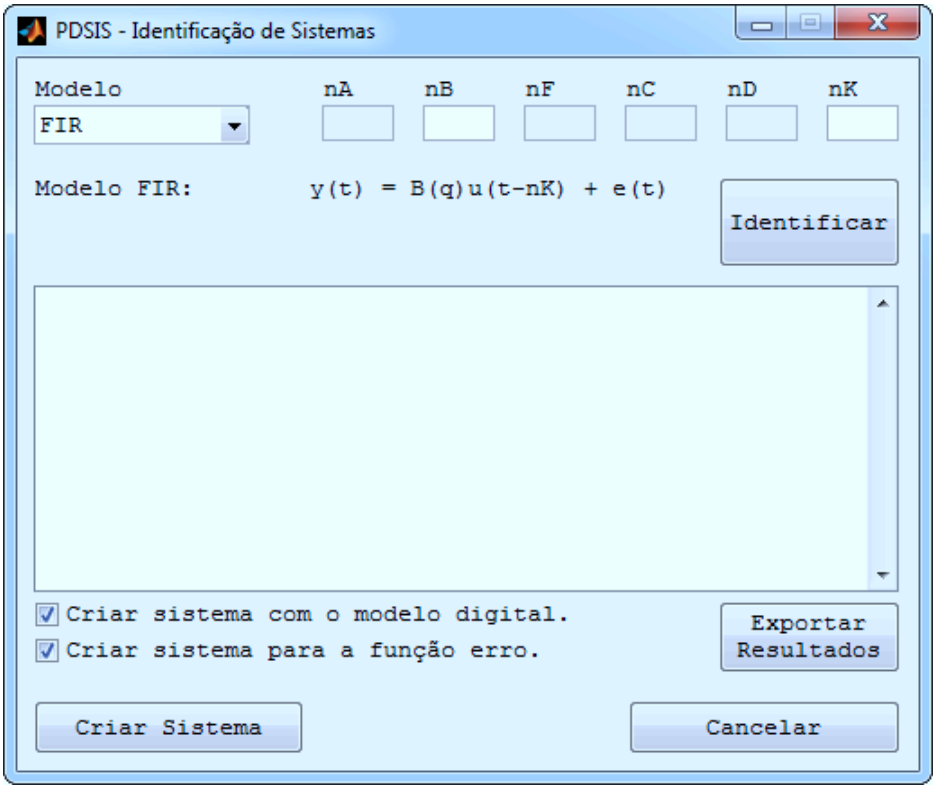


Figura 3.9 – Janela de identificação de sistemas

O PDSIS permite utilizar cinco métodos para a identificação de sistemas: FIR (*Finite Impulse Response*), OE (*Output Error*), ARX (*AutoRegresive with eXternal input*), ARMAX (*AveRage Model with eXternal input*) e BJ (Box-Jenkins). Qualquer um destes modelos podem ser selecionados no campo *Modelo*, no topo esquerdo da janela.

Logo abaixo, é mostrado a equação geral do modelo de identificação selecionado. Os campos *nA*, *nB*, *nC*, *nD* e *nE* permitem configurar a ordem das equações *A*, *B*, *C*, *D* e *E*, respectivamente, e o campo *nK* configura a ordem do atraso na entrada do sistema.

Depois de configurados os parâmetros de identificação, pode-se identificar o sistema clicando no botão *Identificar*, posicionado na parte direita da tela. Este botão executa a função de identificar o sistema selecionado previamente, e gerar um pequeno relatório sobre a identificação, que é exibido na caixa de texto posicionada no meio da janela.

O relatório gerado contém informações sobre a identificação, como o nome do sistema utilizado, o número de amostras, os coeficientes utilizados, qual modelo de identificação foi utilizado, os polinômios (*A*, *B*, *C*, *D* e *E*) encontrados e o erro de previsão final (FPE). Também são geradas as funções de transferência da planta e do erro, nas formas digital e analógica. O PDSIS permite exportar este relatório obtido para um arquivo de texto através do botão *Exportar Resultados*.

Quando o sistema selecionado é identificado, o PDSIS permite que sejam criados sistemas do tipo simulação com a função de transferência encontrada na identificação. Isto permite que o sistema identificado seja simulado e seu comportamento seja analisado para diversos sinais de entrada e controladores. O diagrama da Figura 3.10 mostra graficamente o funcionamento da função de identificação.

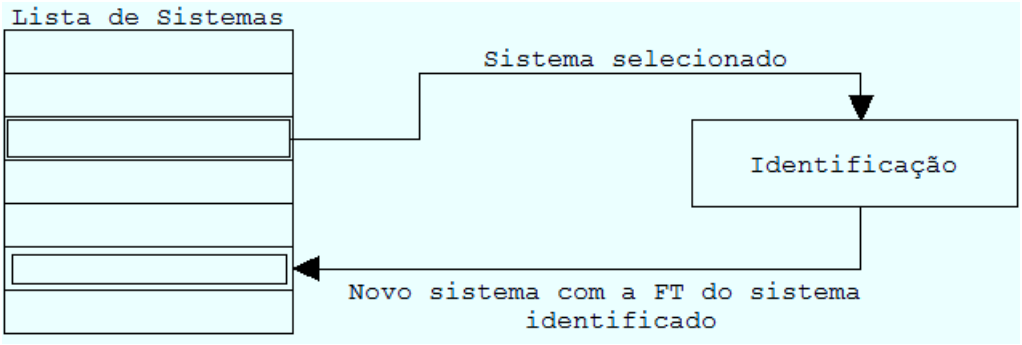


Figura 3.10 – Diagrama da função de identificação

A caixa de seleção *Criar sistema com modelo digital* permite escolher se o sistema será criado com a função de transferência digital ou analógica. A caixa marcada significa criar com a função de transferência digital, a caixa desmarcada significa criar com a função de transferência em sua forma analógica.

Durante a identificação, também é calculado a função de transferência do erro. Quando caixa de seleção *Criar sistema para função erro* está marcada, é criado um sistema do tipo simulação com esta função de transferência.

4 Depuração do PDSIS

A fim de testar o funcionamento do PDSIS, foram escolhidos 3 sistemas com diferentes características, para cobrir todas os possíveis casos de utilização do PDSIS. A Tabela 4.1 descreve o nome, o tipo, a entrada e a planta de cada um.

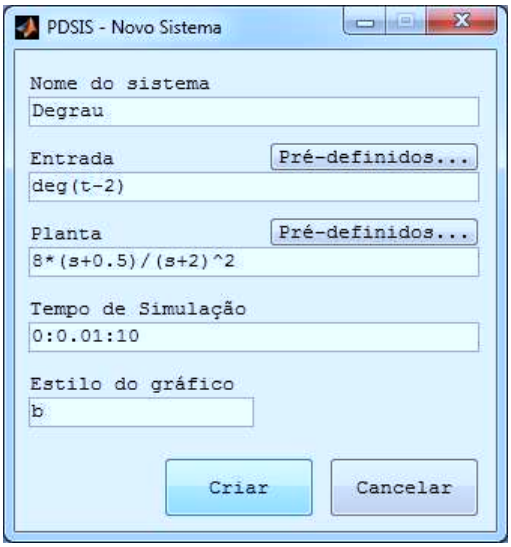
Tabela 4.1 – Descrição dos sistemas utilizados para a realização dos testes.

Nome	Tipo	Entrada	Planta
Degrau	Simulação	$deg(t-2)$	$8*\frac{(s+0{,}5)}{(s+2)^2}$
Seno	Simulação	$sin(8*pi*t)$	$\frac{10*0{,}009901}{z-0{,}9802}$
PRBS	Experimento	$prbs(1)+e(t)$	$\frac{0{,}01z-0{,}01}{z^{52}-1{,}94z^{51}+0{,}94z^{50}}$

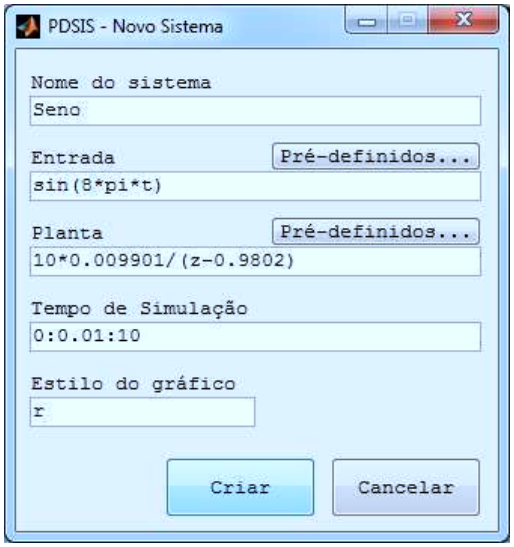
Os 3 sistemas foram simulados para um período de amostragem de 10ms, e tempo total de 10s. O sistema *PRBS* descrito na tabela 4.1 teve seus dados gerados em MATLAB® e salvos em arquivos de texto. A função do sinal de entrada, $prbs(1)$, é somada com um erro aleatório, $e(t)$, gerado pela função `rand()` com uma amplitude de 10% da amplitude do prbs. O código utilizado para gerar os dados experimentais é mostrado no apêndice A.

4.1 Criando os Sitemas

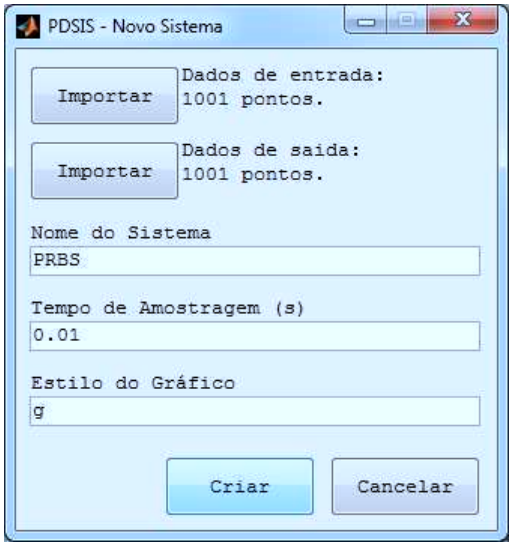
Para a realização dos testes, os sistemas foram criados seguindo os passos descritos na Seção 3.4.1. Os sistemas tiveram seus parâmetros programados de acordo com a Tabela 4.1. A Figura 4.1 mostra as janelas de criação de cada sistema.



(a) Sistema *Degrau*



(b) Sistema *Seno*



(c) Sistema *PRBS*

Figura 4.1 – Janelas de criação dos 3 sistemas.

Com os sistemas criados, o programa retorna à janela principal, mostrando os gráficos do sinal de entrada e de saída e os diagramas de Bode e de Nyquist dos 3 sistemas. A Figura 4.2 mostra a janela principal com os gráficos preenchidos com os dados dos 3 sistemas criados.

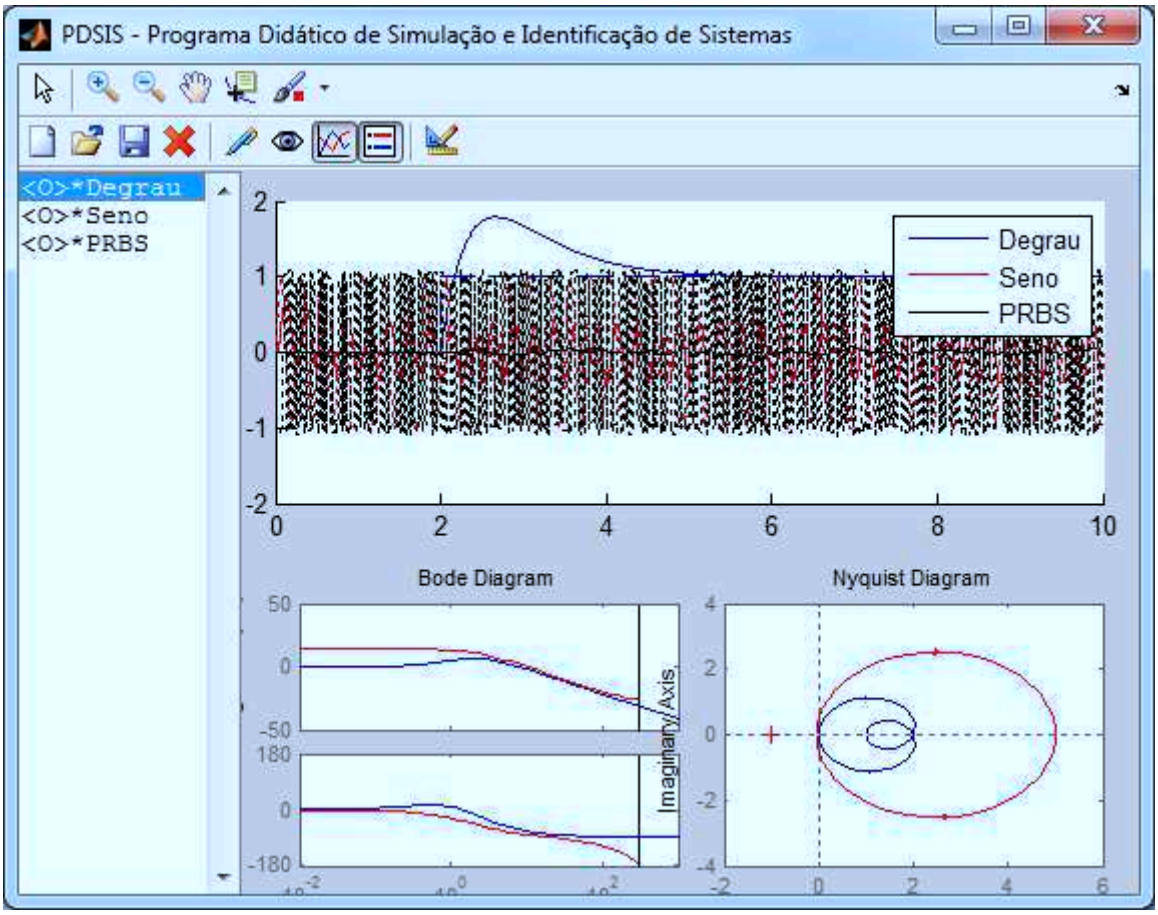


Figura 4.2 – Janela principal após a criação dos 3 sistemas.

A priori, o que é notado na janela principal, após a criação dos sistemas, são os gráficos dos sinais de entrada dos sistemas que poluem o gráfico da resposta no tempo. Então, o gráfico dos sinais de entrada foram escondidos pressionando-se o botão *Esconder/Visualizar Entradas*, a fim de melhorar a visualização dos sinais de resposta dos sistemas no tempo. A Figura 4.3 mostra a janela principal com os sinais de entrada escondidos.

Também nota-se nos diagramas de Bode e Nyquist que o sistema *PRBS* não está presente. Isto era esperado pois, o sistema *PRBS* é composto apenas por dados experimentais e não possui uma planta, pois não foi ainda identificado.

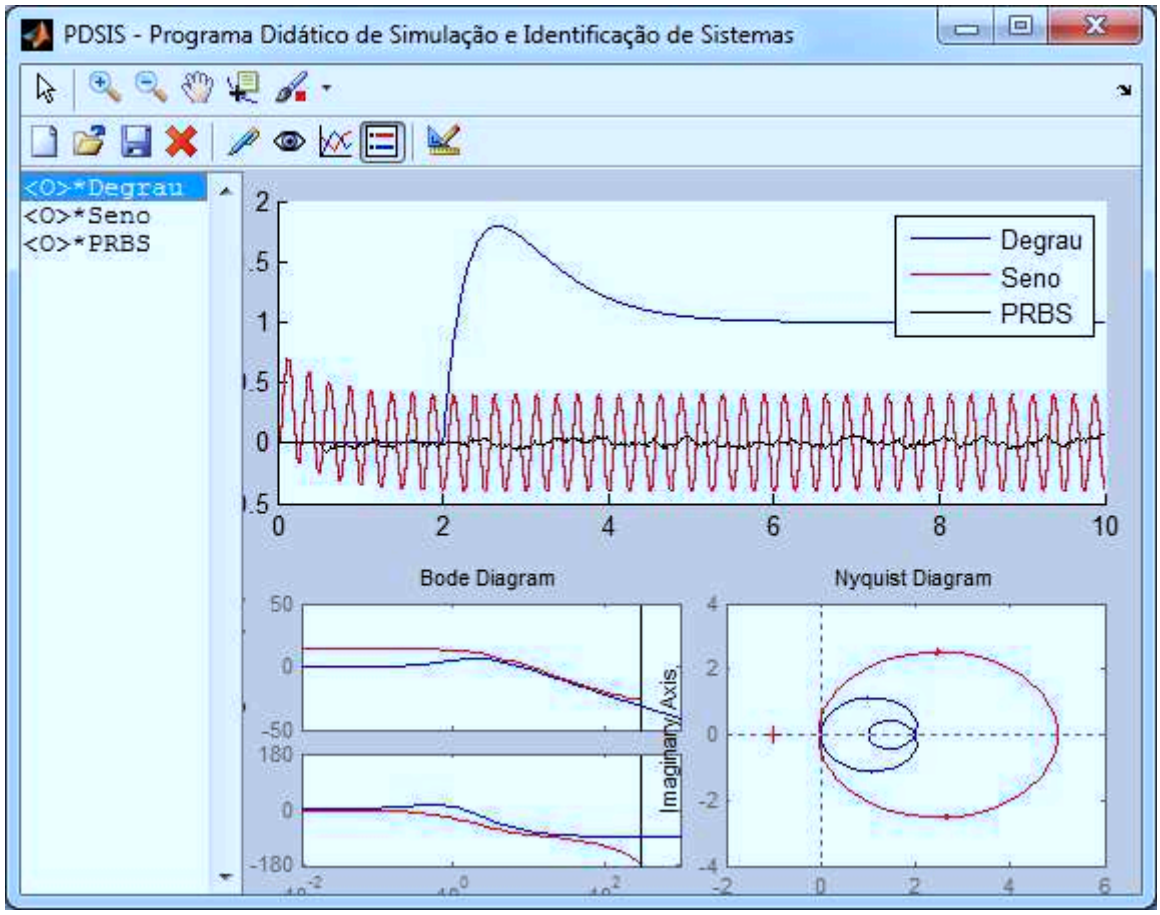


Figura 4.3 – Janela principal com os gráficos dos sinais de entrada escondidos.

4.2 Salvando e Carregando Sistemas em Arquivos

Para testar o salvamento e o carregamento de sistemas em arquivos, os 3 sistemas foram primeiramente salvos em diferentes arquivos e depois excluídos da lista de sistemas. Então, os sistemas foram carregados a partir dos arquivos e inseridos novamente na lista. A Figura 4.4 mostra o resultado obtido após estas ações serem executadas.

Nota-se na Figura 4.4 que o símbolo $\star$, que indica que o sistema foi modificado e não foi salvo, não aparece mais ao lado esquerdo dos nomes dos sistemas. Isto era esperado, dado que os sistemas foram carregados a partir de arquivos, o que indica que já foram salvos, e não foram modificados após isto.

Assim, baseado nos resultados obtidos, pode-se afirmar que o sistema de arquivamento do PDSIS está funcionando como se era esperado.

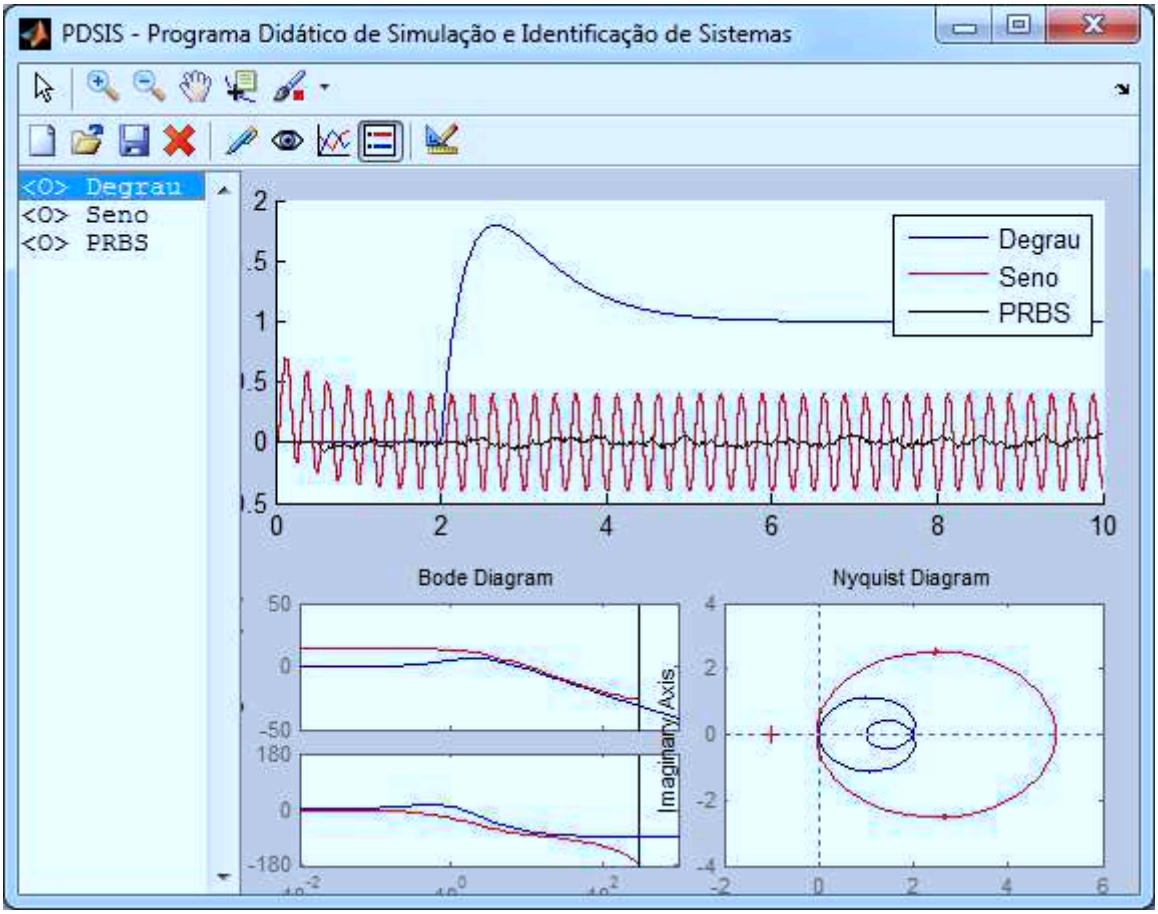


Figura 4.4 – Janela principal com os sistemas carregados a partir de arquivos.

4.3 Identificação dos Sistemas

Para testar a parte de identificação de sistemas do PDSIS, foram escolhidos 3 métodos de identificação, um para cada sistema. Após a identificação de cada sistema, foram analisados os relatórios gerados e os sistemas criados com os resultados das identificações. A Tabela 4.2 mostra os métodos e os coeficientes utilizados para a idenfiticação de cada sistema.

Os coeficientes utilizados foram obtidos na tentativa e erro, até ser produzido um sistema de características similares aos originais baseado no erro de previsão final e na

Sistema	Método	nA	nB	nF	nC	nD	nK
Degrau	OE	-	2	3	-	-	0
Seno	ARMAX	2	3	-	2	-	0
PRBS	ARX	2	2	-	-	-	51

Tabela 4.2 – Tabela de método e coeficientes utilizados para a identificação de cada sistema.

análise dos gráficos. A Tabela 4.3 mostra as plantas originais dos sistemas, as plantas obtidas a partir das identificações e o erro de previsão final (FPE) de cada identificação. O apêndice B contém os relatórios gerados nas identificaões pelo PDSIS.

Sistema	Planta Original	Planta Identificada	FPE
Degrau	$8*\frac{(s+0{,}5)}{(s+2)^2}$	$\frac{0{,}04s^3+7{,}76s^2+543{,}7s+269{,}63}{s^3+712{,}6s^2+273{,}9s+269{,}65}$	$1,95 * 10^{-6}$
Seno	$\frac{10*0{,}009901}{z-0{,}9802}$	$\frac{0{,}00003z^2+0{,}099z+0{,}05}{z^2-0{,}48z-0{,}49}$	$1,19 * 10^{-33}$
PRBS	$\frac{0{,}01z-0{,}01}{z^{52}-1{,}94z^{51}+0{,}94z^{50}}$	$\frac{0{,}01z-0{,}01}{z^{52}-1{,}94z^{51}+0{,}94z^{50}}$	$0,44 * 10^{-12}$

Tabela 4.3 – Tabela de resultados das identificações.

Com os resultados das identificações, foram criados 3 novos sistemas: *Degrau (OE)* é um sistema criado com a planta analógica resultante da identificação de *Degrau*; *Seno (ARMAX)* é um sistema criado com a planta digital resultante da identificação de *Seno*; e *PRBS (ARX)* é um sistema criado com a planta digital resultante da identificação de *PRBS*.

As Figuras 4.5, 4.6 e 4.7 mostram os seis sistemas na lista e os gráficos dos sistemas *Degrau* e *Degrau (OE)*, *Seno* e *Seno (ARMAX)* e *PRBS* e *PRBS (ARX)*, respectivamente, com os demais sistemas omitidos.

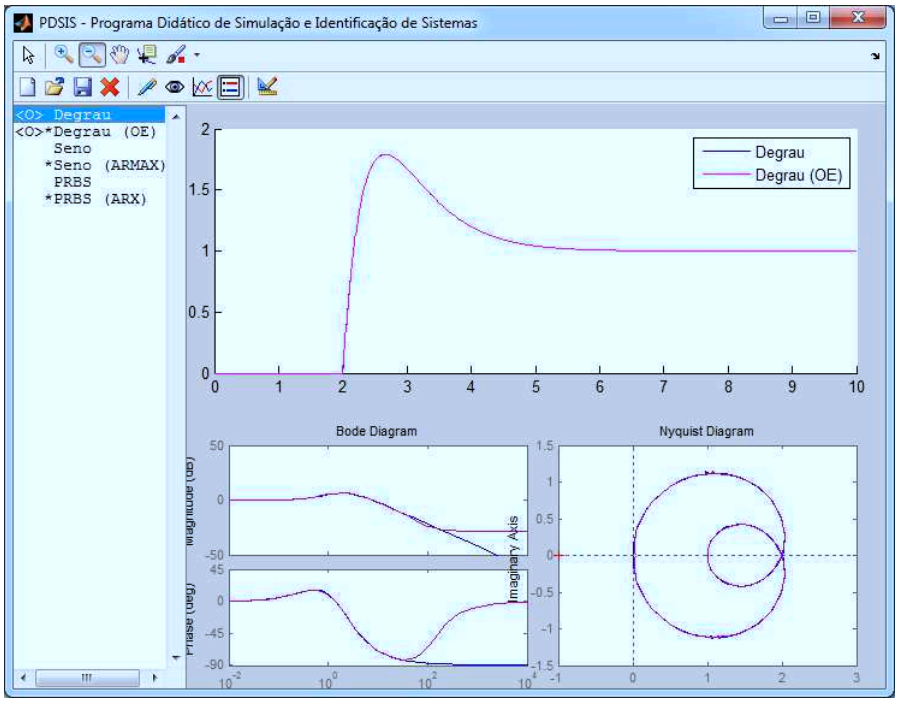


Figura 4.5 – Janela principal com os gráficos de comparação entre *Degrau* e *Degrau (OE)*.

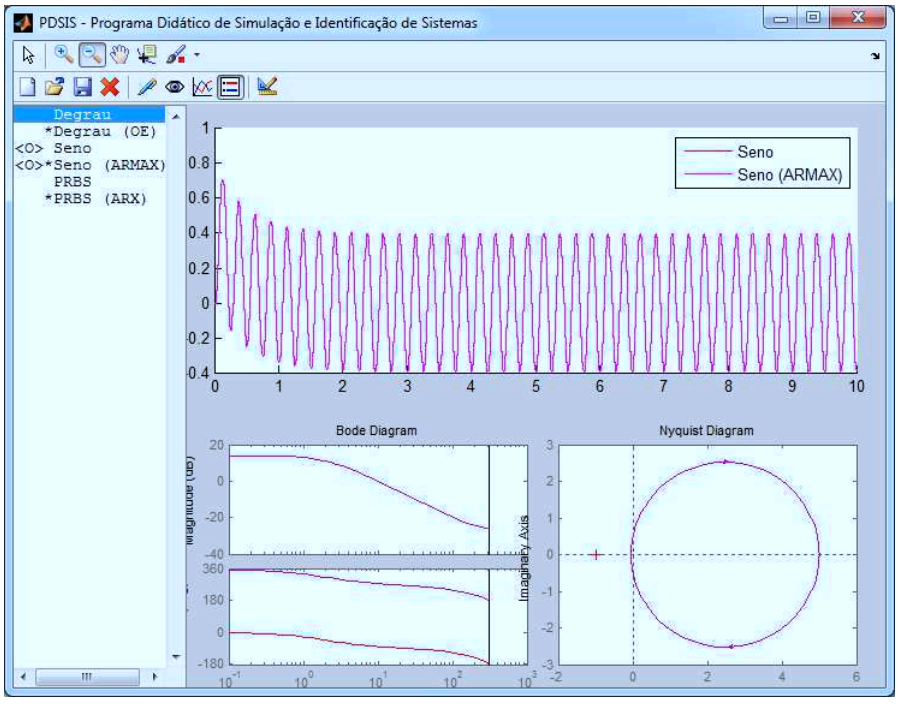


Figura 4.6 – Janela principal com os gráficos de comparação entre *Seno* e *Seno (ARMAX)*.

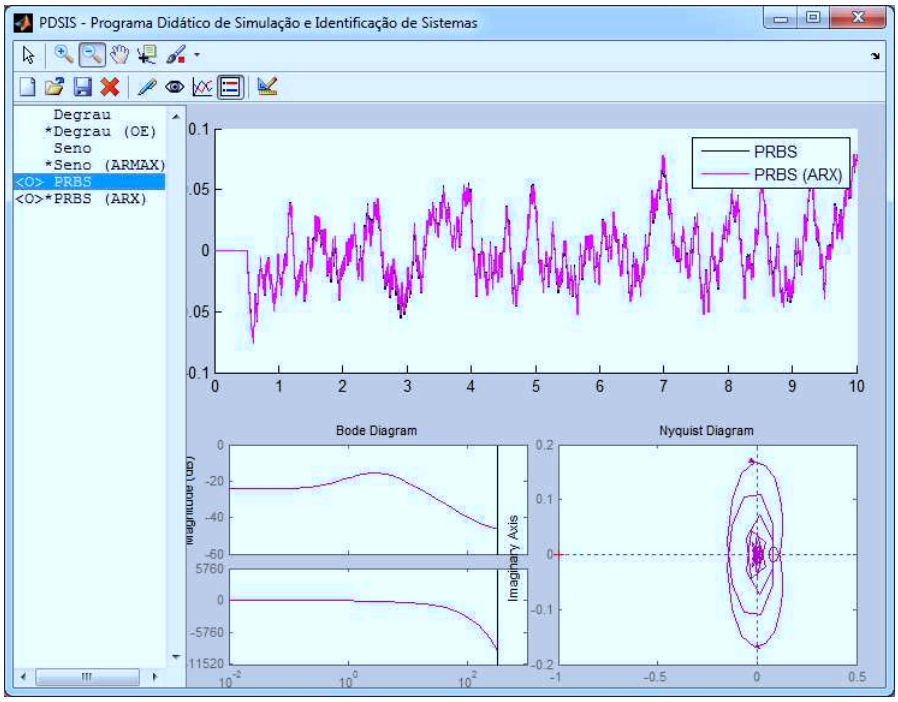
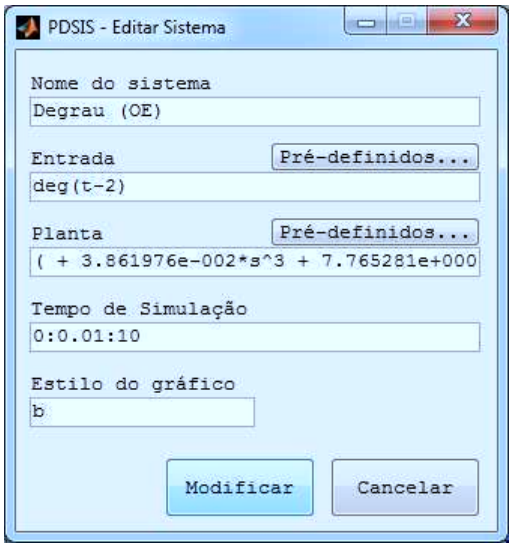


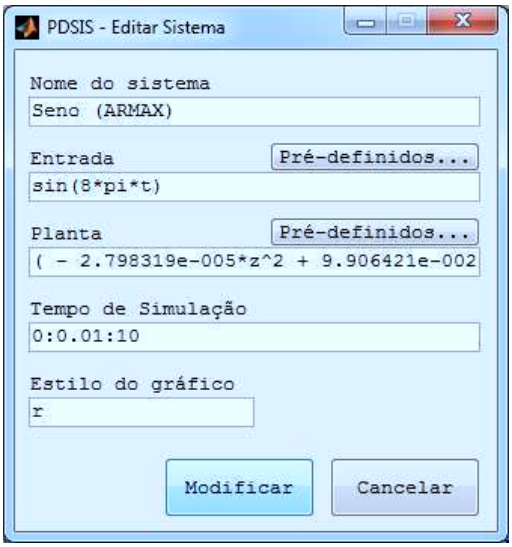
Figura 4.7 – Janela principal com os gráficos de comparação entre *PRBS* e *PRBS (ARX)*.

A Figura 4.8 mostra as janelas de edição dos sistemas *Degrau (OE)*, *Seno (ARMAX)* e *PRBS (ARX)* com todos os dados do sistema original, exceto pelo campo *Planta*, que foi preenchido com o resultado da identificação. Apenas os campos *Entrada* e *Estilo* foram modificados, já que por padrão, o PDSIS coloca a entrada do novo sistema como sendo um degrau e o estilo como o estilo do sistema original.

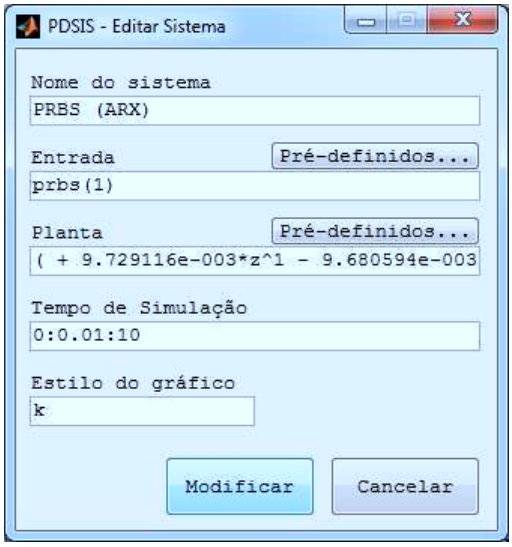
Assim, baseado nos resultados obtidos, pode-se afirmar que o PDSIS realizou, com sucesso, as tarefas de identificação, geração dos relatórios e criação dos novos sistemas.



(a) Janela de edição de *Degrau (OE)*



(b) Janela de edição de *Seno (ARMAX)*



(c) Janela de edição de *PRBS (ARX)*

Figura 4.8 – Janelas de edição dos 3 sistemas gerados pelas identificações.

5 Conclusão

O projeto do PDSIS conseguiu manter uma interface limpa e simples, e incluir diversas ferramentas de simulação e identificação, sem comprometer a facilidade e agilidade durante sua execução.

Os testes realizados foram capazes de verificar o funcionamento de todas as ferramentas disponíveis no PDSIS: criação e edição de sistemas, exclusão de sistemas, manipulação de arquivos, omissão/visualização dos sinais de entrada, omissão/visualização de sistemas selecionados, identificação, geração de relatórios e criação de sistemas a partir dos resultados obtidos da identificação. Em nenhuma das ferramentas do PDSIS, foram encontrados problemas ou erros, comprovando sua estabilidade e solidez para o uso didático.

Assim, pode-se afirmar que o projeto do PDSIS foi realizado com sucesso, atingindo o objetivo de ser simples e funcional para uso didático. Além disso, o código simples e o uso de apenas uma classe durante todo o projeto é um atrativo para que o programa seja editado e melhorado futuramente.

6 Trabalhos Futuros

Apesar do PDSIS ter conseguido alcançar todos os objetivos, há diversas áreas em que ele pode ser melhorado.

Para futuras versões, a área de gráficos poderá conter outros gráficos complementares, como o espectro de potência e o lugar das raízes. Poderia ser dado a escolha ao usuário sobre o *layout* dos gráficos como, por exemplo, ao invés de um gráfico na parte superior, poderia ser escolhido mostrar dois gráficos. Também poderia ser escolhido quais gráficos seriam mostrados e em qual janela, tendo o usuário o poder de omitir os gráficos desinteressantes para ele. Para tais mudanças, a janela principal deve ter seus 3 gráficos (*axes*) trocados por apenas um utilizando *subplots*.

Na parte de identificação de sistemas, é esperado que, futuramente, haja gráficos auxiliares na janela de indentificação com mais informações da identificação, permitindo ao usuário observar o comportamento da planta encontrada para a mesma entrada do sistema original e comparar os resultados rapidamente sem a necessidade de sair do ambiente de identificação.

É também esperado para futuras versões que o PDSIS incluam mais funções como, por exemplo, a possibilidade de incluir um controlador à planta e permitir que esta seja simulada em malha fechada ou em malha aberta.

Em questão de código, a classe *sistema* será melhorada para se adaptar às necessidades das novas versões, sendo natural o uso de hierarquia de classes, ao invés de apenas uma.

7 Bibliografia

GUZMÁN, J. L.; RIVERA, D. E.; DORMIDO, S.; BERENGUEL, M. An interactive software tool for system identification. *Science Direct*: Volume 45, edição 1, p. 115-123, Mar. 2012.

LENNART, Ljung.*System Identification*. Linköping: Department of Electrical Engineering, 1995.

ATTAWAY, Stormy. *MATLAB® :* A practical Introduction to Programming and Problem Solving. 2 Ed. Boston: Butterworth-Heinemann, 2012.

HAHN, Brian D.; VALENTINE, Daniel T. *Essential MATLAB for Engineers and Scientists*. 3 Ed. Cape Town: Butterworth-Heinemann, 2007.

CHAPMAN, Stephen J. *MATLAB® Programming for Engineers*. 4 Ed. Southbank: Thomson, 2008.

A Código de Geração de Dados Experimentais

```
t = [0:0.01:10];
comp = length(t);
entrada = idinput(comp,'prbs')+.1*(1-2*rand(comp,1));
s = tf('s');
planta = c2d(exp(-.5*s)*(s+.5)/((s+2)*(s+4)),0.01);
saida = lsim(planta, entrada, t);
f = fopen('c:/entrada.txt','w');
fprintf(f, '%f\n',entrada);
fclose(f);
f = fopen('c:/saida.txt','w');
fprintf(f, '%f\n',saida);
fclose(f);
```

B Relatórios de Identificação

B.1 Relatório - Sistema *Degrau*

```
PDSIS - Identificação de Sistemas
_____________________________________

Modelo utilizado: OE
nB = 2 nF = 3 nK = 0

*************************

Sistema: Degrau (1001 pontos), Amostragem: 0.01s

*************************

Polinômios encontrados:

B(q) = + 0.038620 - 3.842737e-002q^-1

F(q) = + 1.000000 - 2.470721e+000q^-1 + 1.961284e+000q^-2 - 4.903707e-001q^-3

*************************

Erro de Previsão Final (FPE):1.9472e-006

*************************

Função de Transferência Discreta:

+ 3.861976e-002z^3 - 3.842737e-002z^2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000z^3 - 2.470721e+000z^2 + 1.961284e+000z^1 - 4.903707e-001z^0

Erro Discreto:

+ 1.391963e-003z^0
- - - - - - - - - -
+ 1.000000e+000z^0

*************************
```

```
Função de Transferência Contínua:

+ 3.861976e-002s^3 + 7.765281e+000s^2 + 5.437248e+002s^1 + 2.696328e+002s^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000s^3 + 7.125936e+001s^2 + 2.738728e+002s^1 + 2.696542e+002s^0

Erro Contínuo:

+ 1.391963e-003s^0
- - - - - - - - - -
+ 1.000000e+000s^0
```

B.2 Relatório - Sistema *Seno*

```
PDSIS - Identificação de Sistemas
_____________________________________

Modelo utilizado: ARMAX
nA = 2 nB = 3 nC = 2 nK = 0

*************************

Sistema: Seno (1001 pontos), Amostragem: 0.01s

*************************

Polinômios encontrados:

A(q) = + 1.000000 - 4.827914e-001q^-1 - 4.875600e-001q^-2

B(q) = - 0.000028 + 9.906421e-002q^-1 + 4.922045e-002q^-2

C(q) = + 1.000000 - 1.976681e-001q^-1 + 5.187805e-001q^-2

*************************

Erro de Previsão Final (FPE):1.1935e-033

*************************
```

```
Função de Transferência Discreta:

- 2.798319e-005z^2 + 9.906421e-002z^1 + 4.922045e-002z^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000z^2 - 4.827914e-001z^1 - 4.875600e-001z^0

Erro Discreto:

+ 3.442904e-017z^2 - 6.805524e-018z^1 + 1.786112e-017z^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000z^2 - 4.827914e-001z^1 - 4.875600e-001z^0

*************************

Função de Transferência Contínua:

- 2.798319e-005s^3 + 9.998203e+000s^2 + 1.396711e+003s^1 + 1.035751e+006s^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000s^3 + 1.416685e+002s^2 + 1.038522e+005s^1 + 2.071317e+005s^0

Erro Contínuo:

+ 3.442904e-017s^3 + 6.964155e-015s^2 + 2.603000e-012s^1 + 3.177648e-010s^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000s^3 + 1.416685e+002s^2 + 1.038522e+005s^1 + 2.071317e+005s^0
```

B.3 Relatório - Sistema *PRBS*

```
PDSIS - Identificação de Sistemas
_____________________________________

Modelo utilizado: ARX
nA = 2 nB = 2 nK = 51

*************************

Sistema: PRBS (1001 pontos), Amostragem: 0.01s
```

```
*************************

Polinômios encontrados:

A(q) = + 1.000000 - 1.940987e+000q^-1 + 9.417630e-001q^-2

B(q) = + 9.729116e-003q^-51 - 9.680594e-003q^-52

*************************

Erro de Previsão Final (FPE):4.383e-013

*************************

Função de Transferência Discreta:

+ 9.729116e-003z^1 - 9.680594e-003z^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000z^52 - 1.940987e+000z^51 + 9.417630e-001z^50

Erro Discreto:

+ 6.786674e-007z^2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000z^2 - 1.940987e+000z^1 + 9.417630e-001z^0

*************************

Função de Transferência Contínua:

«OMITIDO»

Erro Contínuo:

+ 1.747904e-007s^2 + 6.991616e-005s^1 + 6.991616e-003s^0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ 1.000000e+000s^2 + 5.999562e+000s^1 + 7.997737e+000s^0
```