apêndice

O *toolbox* DTMJLS para uso com MATLAB

1 Guia do usuário

1.1 Introdução

O *toolbox* DTMJLS (de *Discrete-Time Markovian Jump Linear Systems*) contém uma série de rotinas para síntese e análise de controladores para sistemas lineares discretos sujeitos a saltos markovianos. As rotinas são de fácil utilização, requerendo um mínimo de pré-processamento dos dados para que elas sejam utilizadas.

1.2 Requisitos do sistema

Os seguintes programas devem estar presentes para utilização do *toolbox*:

- Matlab 4.x ou superior;
- LMI Control Toolbox 1.0 ou superior.

Além disso se recomenda que para resolver problemas de ordem muito grande ou para realizar simulações de Montecarlo de maneira intensiva o sistema tenha bastante memória disponível.

O *toolbox* DTMJLS funciona em qualquer plataforma para a qual sejam disponíveis tanto o Matlab como o LMI Control Toolbox.

1.3 Estruturas de dados

Quando se trata com sistemas markovianos, é necessário lidar com N-uplas de matrizes. Tais estruturas de dados são manipuladas por este *toolbox* após serem devidamente convertidas em vetores. Isso é feito de maneira quase transparente ao usuário por meio de dois comandos: `stack` e `unstack`. Por exemplo para se criar uma N-upla contendo duas matrizes A_1 e A_2 basta digitar a seguinte sequência de comandos:

```
A1 = [1 2; 3 4]; B1 = [5 6; 7 8];
Avec = stack(A1,A2);
```

Com isso é criada uma variável `Avec` que contém a N-upla (A_1, A_2). Para obter os elementos a partir de uma N-upla já existente, pode se utilizar do comando `unstack` de diferentes maneiras, como

```
[X1,X2] = unstack(Avec);
```

que atribui às variáveis `X1` e `X2` os valores dos elementos 1 e 2 da N-upla `Avec`, ou

```
X1 = unstack(Avec,1);
X2 = unstack(Avec,2);
```

que extrai componentes individuais de `Avec`.

Uma N-upla armazenada dessa forma pode ter a princípio quantos elementos quanto se queira, respeitadas as limitações do sistema. Todas as rotinas do *toolbox* DTMJLS usam essa forma para receber ou fornecer argumentos que sejam N-uplas de matrizes, vetores ou escalares.

1.4 Um exemplo simples de aplicação

O seguinte exemplo mostra como deve ser a operacionalização do uso das rotinas do *toolbox* DTMJLS. Seja o seguinte problema de controle ótimo: Obter F tal que $u_{\theta(k)} = F_{\theta(k)}x(k)$ minimize o seguinte funcional:

$$J(x(0), \theta(0), u) = \frac{1}{2}\sum_{k=0}^{\infty} E\left\{ \begin{bmatrix} x(k)^* & u(k)^* \end{bmatrix} \begin{bmatrix} Q_{\theta(k)} & L_{\theta(k)} \\ L^*_{\theta(k)} & R_{\theta(k)} \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \right\}$$

sujeito a

$$x(k+1) = A_{\theta(k)}x(k) + B_{\theta(k)}u(k)$$
$$x(0) = x_0 \qquad \theta(0) = \theta_0$$

onde $\theta(k)$ é uma cadeia de Markov assumindo valores no conjunto $\{1, 2\}$ baseada na matriz de probabilidades de transição P. A seguinte sequência de comandos cria as matrizes necessárias para a definição do problema:

```
A1=0.8353;   A2=0.9646;
B1=0.0915;   B2=0.0982;
Q1=0.0355;   Q2=Q1;
R1=1;        R2=R1;
L1=0;        L2=L1;
P=[0.9767 0.0233; 0.0435 0.9565];
```

e a seguinte sequência cria as N-uplas:

```
Avec=stack(A1,A2); Bvec=stack(B1,B2);
Qvec=stack(Q1,Q2); Lvec=stack(L1,L2); Rvec=stack(R1,R2);
```

O problema de controle ótimo propriamente é resolvido com o uso da função `mkvlqr`.

```
[Fvec,Xvec] = mkvlqr(Avec,Bvec,Qvec,Lvec,Rvec,P);
```

Os valores fornecidos `Fvec`, lei de controle ótima e `Xvec`, solução das equações algébricas de Riccati acopladas associadas ao problema de controle (ambas N-uplas) podem ser descompactas por meio de

```
[X1,X2]=unstack(Xvec);
[F1,F2]=unstack(Fvec);
```

Para o caso deste exemplo, o resultado fornecido pelo Matlab é:

```
F1 =
   -0.0103
F2 =
   -0.0331
```

De modo geral, todos as funções do toolbox DTMJLS têm uso mais ou menos semelhante a este, com sequências de comandos `stack` antes da execução para compactar os argumentos da função e sequências de comandos `unstack` após a execução para descompactar o resultado.

Neste manual as variáveis que contém N-uplas usualmente recebem o sufixo `-vec`. e.g. `Avec`, `Bvec`, etc.

1.5 Créditos

Este toolbox foi escrito por R. P. Marques como parte integrante da tese de doutoramento “Algoritmos de Controle para Sistemas Sujeitos a Saltos Markovianos” apresentada à Escola Politécnica

da Universidade de São Paulo. As rotinas `stack` e `unstack` contudo são de domínio público e fazem parte do pacote de programação convexa LMITOOL, escrito por L. El Ghaoui, R. Nikoukha e F. Delebecque (disponível pela Internet no endereço ftp://ftp.ensta.fr/pub/elghaoui/lmitool).

2 Guia de referência

Esta seção contém informações detalhadas de todas as funções do toolbox DTMJLS. Inicialmente é apresentada uma lista de funções agrupadas por assunto e continua com uma lista em ordem alfabética.

Projeto e análise de controladores	
`hinfopt`	Projeto H_∞ ótimo (abordagem convexa, problema aproximado).
`hinfsopt`	Projeto H_∞ sub-ótimo (algoritmo recursivo).
`mixopt`	Projeto misto H_2/H_∞ (abordagem convexa, problema aproximado).
`mkvlqr`	Regulador linear-quadrático.
`solvhinf`	Projeto H_∞ ótimo (algoritmo misto, cond. inicial dada por `hinfopt`).
`solvhin2`	Projeto H_∞ ótimo (algoritmo misto, condição inicial fornecida).
`specrad`	$r_\sigma(\mathcal{L})$, raio espectral do operador .

Simulação de sistemas	
`l2sim`	Simulação de sistemas sujeitos a perturbações l_2.
`lqrsim`	Simulação de sistemas com (não necessariamente) regulador LQ.
`mcl2sim`	Simulações de Montecarlo com `l2sim`.
`mclqrsim`	Simulações de Montecarlo com `lqrsim`.

Utilidades	
`cloopvec`	Fecha a malha de sistemas.
`eop`	Operador valor esperado.
`maxcare`	Solução máxima para as equações algébricas de Riccati acopladas.
`mkvchain`	Realização de cadeias de Markov.
`stack`	Compactação de N-uplas.
`unstack`	Descompactação de N-uplas.

cloopvec

Propósito

Fecha a malha de sistemas descritos por N-uplas de matrizes.

Sinopse

```
Gvec = cloopvec(Avec,Bvec,Fvec)
```

Descrição

`cloopvec` obtém a descrição do sistema em malha fechada tal que para $A = (A_1, \ldots, A_N)$, $B = (B_1, \ldots, B_N)$ e $F = (F_1, \ldots, F_N)$, $G = (G_1, \ldots, G_N)$ é tal que $G_i = A_i + B_i F_i$ para $i = 1, \ldots, N$.

Obviamente as dimensões de A, B e F devem ser compatíveis entre si.

eop

Propósito

Operador $\mathcal{E}(.)$ (valor esperado).

Sinopse

```
Evec = eop(Xvec,P)
```

Descrição

Para $X = (X_1, \ldots, X_N)$, $\mathcal{E}(X) = (\mathcal{E}_1(X), \ldots, \mathcal{E}_N(X))$ é tal que $\mathcal{E}_i(X) := \sum_{j=1}^{N} p_{ij} X_j$ para $i = 1, \ldots, N$, onde $P = [p_{ij}]$ é a matriz de transição de probabilidades.

Note que `Evec` e `Xvec` são N-uplas. `Evec` pode ser descompactado pelo comando `unstack`.

hinfopt

Propósito

Projeto H_∞ ótimo via programação convexa para um problema aproximado.

Sinopse

```
Fvec = hinfopt(Avec,Bvec,D,Mvec,Jvec,P)
Fvec = hinfopt(Avec,Bvec,D,Mvec,Jvec,P,opt,v)
[Fvec,nu,Qvec,Lvec,Yvec] = hinfopt(...)
```
etc.

Descrição

`hinfopt` resolve o problema H_∞ via programação convexa (veja [1]). `Avec`, `Bvec`, `D`, `Mvec`, e `Jvec` contêm a descrição do sistema, `P` é a matriz de probabilidades de transição, `opt` contém opções para o algoritmo de otimização (digite `help mincx` para maiores informações ou consulte o manual do LMI Control Toolbox) e `v` controla a apresentação na tela de informações sobre a construção do problema de otimização (`v=0` significa "sem informações").

`Fvec` contém o controlador, `nu` é o valor mínimo para o limite superior da norma H_∞ e `Qvec`, `Lvec` e `Yvec` são N-uplas associadas ao problema de otimização.

Note que a solução obtida por `hinfopt` pode ser conservadora, porém outros métodos podem ser utilizados em alguns casos para comparação.

Alternativamente `P`, a matriz de probabilidades de transição pode ser substituída por uma N-upla `Pvec`, contendo um conjunto convexo (`Pvec=stack(P1,P2,...);`) para projeto com estabilização robusta.

Limitações

`D` deve ser uma matriz (N-uplas não são aceitas).

Algoritmo

`hinfsopt` utiliza o algoritmo `mincx` do LMI Control Toolbox.

Veja também

`hinfsopt`, `solvhinf`, `solvhin2`, `mincx`

Referências

[1] COSTA, O. L. V.; MARQUES, R. P. Mixed H_2/H_∞-control of discrete-time Markovian jump linear systems. *IEEE Transactions on Automatic Control*, (a ser publicado).

hinfsopt

Propósito

Projeto H_∞ sub-ótimo, feito por meio de um algoritmo recursivo.

Sinopse

```
[Fvec,Xvec] = hinfsopt(Avec,Bvec,Dvec,Mvec,Jvec,P,nu)
[Fvec,Xvec] = hinfsopt(Avec,...,P,nu,opt1,opt2)
```

Descrição

`hinfsopt` resolve o problema de controle H_∞ sub-ótimo (isto é, dado um valor de `nu`, obter um controlador tal que a norma H_∞ do sistema seja menor que `nu`) via um algoritmo recursivo (veja [1]). Inicialmente é obtida uma solução inicial para um conjunto específico de equações algébricas de Riccati acopladas (EARA) (resolvidas via programação convexa), utilizada como ponto de partida para a parte iterativa do algoritmo.

`Avec`, `Bvec`, `Dvec`, `Mvec`, `Jvec` contêm a descrição do sistema, `P` é a matriz de probabilidades de transição e `nu` o limite superior para a norma H_∞ do sistema. `opt1` contém opções para o algoritmo de otimização que resolve as EARA iniciais (digite `help mincx` para mais informações), e `opt2` é um vetor de três elementos que contém opções para a parte recursiva do algoritmo:

`opt2(1)`: Tolerância relativa para convergência (*default* 1×10^{-3})

`opt2(2)`: Número máximo de iterações (*default* 200)

`opt2(3)`: Exibição de informações na tela (*default* 0 = "sem informações").

`Fvec` contém o controlador calculado e `Xvec` contém a solução para as EARA associadas ao problema de controle H_∞.

`hinfsopt` para com erro caso o problema não tenha solução. Há uma sintaxe alternativa para uso em processamento de lote, não discutida aqui (use `type hinfsopt` para obter informações adicionais).

Algoritmo

`hinfsopt` utiliza o algoritmo `mincx` do LMI Control Toolbox para resolver as EARA iniciais e o algoritmo recursivo apresentado em [1] para obter a solução final.

Veja também

`hinfopt`, `solvhinf`, `solvhin2`, `mincx`

Referências

[1] COSTA, O. L. V.; MARQUES, R. P. *Maximal and stabilizing hermitian solutions for discrete-time coupled algebraic Riccati equations.* (a ser publicado)

l2sim

Propósito

Simulação da resposta de um sistema em malha fechada sujeito a perturbações l_2.

Sinopse

```
[Z,X,W,C] = l2sim(Avec,Bvec,Fvec,Dvec,Mvec,Jvec,...
                  P,x0,th0,n,wopt,s)
```

Descrição

l2sim realiza a simulação do seguinte sistema:

$$\Sigma = \begin{cases} x(k+1) = (A_{\theta(k)} + B_{\theta(k)}F_{\theta(k)})x(k) + D_{\theta(k)}\omega(k) \\ z(k) = M_{\theta(k)}x(k) + J_{\theta(k)}u(k) \\ x(0) = x_0 \qquad \theta(0) = \theta_0 \end{cases}$$

com $\theta(k)$, valor assumido por uma cadeia de Markov com matriz de probabilidades de transição P.

`Avec`, `Bvec`, `Fvec`, `Dvec`, `Mvec`, `Jvec` contêm a descrição do sistema, `x0` e `th0` são as condições iniciais do estado do sistema e da cadeia de Markov, `n` é o número de pontos da simulação, `wopt` (veja abaixo) contém informações sobre a perturbação l_2 e `s` (opcional) é a semente para o gerador de números aleatórios da realização da cadeia de Markov.

`Z`, `X` contém o histórico da saída e do estado do sistema, `W` contém o histórico da perturbação gerada e `C` contém a realização da cadeia de Markov.

As perturbações aplicadas (nesta versão. Em versões futuras pretende-se ampliar esse recurso) obedecem à seguinte equação:

$$\omega(k) = \alpha^k \begin{bmatrix} \sigma_1(k) \\ \vdots \\ \sigma_r(k) \end{bmatrix},$$

onde α = `wopt(1)` e $\sigma_1(k)$, a $\sigma_r(k)$ são variáveis aleatórias independentes com distribuição normal de média nula e desvio padrão `wopt(2)`. O terceiro elemento de `wopt` (opcional) contém a semente para o gerador da perturbação (independente do gerador da

sequência aleatória de estados da cadeia de Markov).

Para não considerar perturbações basta fazer `wopt(1)=0`.

Veja também

`lqrsim`, `mcl2sim`, `mclqrsim`

lqrsim

Propósito

Simulação de um sistema em malha fechada com um regulador linear-quadrático.

Sinopse

```
[X,J,C] = lqrsim(Avec,Bvec,Fvec,Qvec,Lvec,Rvec,P,x0,th0,n,s)
```

Descrição

lqrsim simula o seguinte sistema:

$$x(k+1) = (A_{\theta(k)} + B_{\theta(k)}F_{\theta(k)})x(k)$$
$$x(0) = x_0 \qquad \theta(0) = \theta_0$$

onde $\theta(k)$ é o valor assumido pela cadeia de Markov no instante k.

`Avec`, `Bvec`, `Fvec` contêm a descrição do sistema. `Qvec`, `Lvec`, `Rvec` contêm a descrição do custo quadrático do controle, utilizada para calcular o vetor `J`. `P` é a matriz de probabilidades de transição, `x0` e `th0` são as condições iniciais do estado e da cadeia de Markov. `n` é o número de pontos da simulação e `s` (opcional) é a semente para o gerador de números aleatórios que dá origem à realização `C` da cadeia de Markov.

`X` contém o histórico do estado do sistema.

Veja também

`l2sim`, `mcl2sim`, `mclqrsim`

maxcare

Propósito

Obtém a solução máxima para as equações algébricas de Riccati acopladas

Sinopse

```
Xvec = maxcare(Avec,Bvec,Qvec,Lvec,Rvec,P,opt,v)
```

Descrição

`maxcare` obtém a solução máxima das EARA via programação convexa, conforme a abordagem de [1]. `Avec`, `Bvec`, `Qvec`, `Lvec`, `Rvec` contém os dados do problema, `P` é a matriz de probabilidades de transição, `opt` contém opções para o algoritmo de otimização (digite `help mincx` para maiores informações) e `v` controla a exibição de informações sobre a construção do problema de otimização.

`Xvec` contém a solução máxima das EARA.

Algoritmo

`maxcare` utiliza o algoritmo `mincx` do LMI Control Toolbox.

Veja também

`mkvlqr`, `mincx`

Referências

[1] COSTA, O. L. V.; MARQUES, R. P. *Maximal and stabilizing hermitian solutions for discrete-time coupled algebraic Riccati equations.* (a ser publicado)

mcl2sim

Propósito

Simulador de Montecarlo para l2sim.

Sinopse

```
[Zvec,Xvec,Wvec,C] = mcl2sim(Avec,Bvec,Fvec,Dvec,Mvec,...
                             Jvec,P,x0,th0,n,nsim,wopt,s)
```

Descrição

`mcl2sim` realiza `nsim` simulações utilizando a função `l2sim` e armazena os resultados em N-uplas. Os argumentos são os mesmos de `l2sim`, a menos de `nsim` que é o número de simulações desejadas. Note que `Zvec`, `Xvec` e `Wvec` são N-uplas, descompactáveis via `unstack`.

Veja o comando `l2sim` para maiores detalhes.

Veja também

`l2sim`

mclqrsim

Propósito

Simulador de Montecarlo para lqrsim.

Sinopse

```
[Xvec,J,C] = mclqrsim(Avec,Bvec,Fvec,Qvec,Lvec,Rvec,...
                      P,x0,th0,n,nsim,s)
```

Descrição

`mclqrsim` realiza `nsim` simulações utilizando a função `lqrsim` e armazena os resultados em N-uplas. Os argumentos são os mesmos de `lqrsim`, a menos de `nsim` que é o número de simulações desejadas. Note que `Xvec` é uma N-upla, que pode ser descompactada via `unstack`.

Veja o comando `lqrsim` para maiores detalhes.

Veja também

`lqrsim`

mixopt

Propósito

Projeto misto H_2/H_∞ via programação convexa para um problema aproximado.

Sinopse

```
Fvec = mixopt(Avec,Bvec,D,Mvec,Jvec,P,nu,opt,v)
[Fvec,upH2,Xvec,Qvec,Lvec,Yvec] = mixopt(...)
```
etc.

Descrição

`mixopt` resolve o problema H_2/H_∞ via programação convexa (veja [1]). `Avec`, `Bvec`, `D`, `Mvec`, e `Jvec` contêm a descrição do sistema, `P` é a matriz de probabilidades de transição, `nu` é o limite superior máximo aceitável para a norma H_∞, `opt` contém opções para o algoritmo de otimização (digite `help mincx` para maiores informações ou consulte o manual do LMI Control Toolbox) e `v` controla a apresentação na tela de informações sobre a construção do problema de otimização (`v=0` significa "sem informações").

`Fvec` contém o controlador, `upH2` é o valor mínimo para o limite superior da norma H_2 e `Xvec`, `Qvec`, `Lvec` e `Yvec` são N-uplas associadas ao problema de otimização.

Note que a solução obtida por `mixopt` pode ser conservadora, e que não há outro método conhecido para comparação.

Alternativamente `P`, a matriz de probabilidades de transição pode ser substituída por uma N-upla `Pvec`, contendo um conjunto convexo (`Pvec=stack(P1,P2,...);`) para projeto com estabilização robusta.

Limitações

`D` deve ser uma matriz (N-uplas não são aceitas).

Algoritmo

`hinfsopt` utiliza o algoritmo `mincx` do LMI Control Toolbox.

Veja também

`mincx`

Referências

[1] COSTA, O. L. V.; MARQUES, R. P. Mixed H_2/H_∞-control of discrete-time Markovian jump linear systems. *IEEE Transactions on Automatic Control*, (a ser publicado).

mkvchain

Propósito

Gera uma realização para uma cadeia de Markov.

Sinopse

```
C = mkvchain(P,ic,n,s)
```

Descrição

`mkvchain` gera uma realização para uma cadeia de Markov definida por sua matriz de probabilidades de transição `P` com estado inicial `ic`. A cadeia assume os estados $1, 2, \ldots, N$, onde N = `dim(P)`. `n` é o número de instantes de tempo considerados na realização e `s` (opcional) é a semente para o gerador de números aleatórios.

Exemplo

O comando

```
chain = mkvchain([0.9 0.1; 0.1 0.9],1,10,1e4);
```

gera como saída a variável `chain`, que é tal que

```
chain =
     1
     1
     2
     2
     2
     2
     2
     2
     2
     2
```

mkvlqr

Propósito

Projeto de regulador linear-quadrático.

Sinopse

```
[Fvec,Xvec] = mkvlqr(Avec,Bvec,Qvec,Lvec,Rvec,P,opt,v)
```

Descrição

`mkvlqr` realiza o projeto do regulador linear-quadrático via programação convexa, conforme a abordagem de [1]. `Avec`, `Bvec`, `Qvec`, `Lvec`, `Rvec` contém os dados da planta e do funcional quadrático, `P` é a matriz de probabilidades de transição, `opt` contém opções para o algoritmo de otimização (digite `help mincx` para maiores informações) e `v` controla a exibição de informações sobre a construção do problema de otimização.

`Fvec` contém o controlador ótimo e `Xvec` contém a solução máxima das EARA associadas.

Algoritmo

`mkvlqr` utiliza o algoritmo `mincx` do LMI Control Toolbox.

Veja também

`maxcare`, `mincx` e a seção 1.4 do guia do usuário

Referências

[1] COSTA, O. L. V.; MARQUES, R. P. *Maximal and stabilizing hermitian solutions for discrete-time coupled algebraic Riccati equations.* (a ser publicado)

solvhinf

Propósito

Projeto H_∞ ótimo via algoritmo misto.

Sinopse

```
[Fvec,nuopt,Xvec] = solvhinf(Avec,Bvec,D,Mvec,Jvec,P,tol)
[Fvec,nuopt,Xvec] = solvhinf(Avec,Bvec,D,Mvec,Jvec,P,tol,...
                             op1,op2,op3,v)
```

Descrição

`solvhinf` combina os algoritmos `hinfopt` e `hinfsopt` para produzir uma solução ótima exata para o problema de controle H_∞. `hinfopt` produz uma solução inicial, refinada iterativamente por `hinfsopt` via bissecção de intervalos.

`Avec,...,Jvec` contêm a descrição do sistema, `P` é a matriz de probabilidades de transição, `tol` é o erro máximo (em termos absolutos) admissível para a solução final, `op1` e `op2` são as opções para a rotina de `hinfsopt` (digite `help hinfsopt` para maiores informações) e `op3` é o vetor de opções para a rotina `hinfopt` (digite `help mincx` ou `help hinfopt` para mais informações). `v` controla a saída de informações sobre o andamento do algoritmo misto (`v=0` significa "sem informações").

`Fvec` contém o controlador desejado, `nuopt` o limite superior para a norma H_∞ e `Xvec` a solução para as EARA associadas ao problema H_∞.

IMPORTANTE: Um bom ajuste dos algoritmos (em especial via `op2`) é essencial para a convergência.

Algoritmo

`solvhinf` utiliza os algoritmos de `hinfopt` e `hinfsopt` combinados segundo o tradicional método da bissecção.

Limitações

`D` deve ser uma matriz (N-uplas não são aceitas).

Veja também

`hinfopt`, `hinfsopt`, `solvhin2`

solvhin2

Propósito

Projeto H_∞ ótimo via algoritmo misto (alternativo a solvhinf).

Sinopse

```
[Fvec,nuopt,Xvec]=solvhin2(Avec,Bvec,D,Mvec,Jvec,P,nu0,tol)
[Fvec,nuopt,Xvec]=solvhin2(Avec,Bvec,Dvec,Mvec,Jvec,P,...
                           nu0,tol,op1,op2,v)
```

Descrição

solvhinf2 é uma rotina alternativa a solvhinf. ela difere desta última no sentido de que a primeira aproximação para o limite superior não é fornecido por hinfsopt, mas sim pelo usuário, o que permite que D seja substituído pela N-upla Dvec.

`Avec,...,Jvec` contêm a descrição do sistema, `P` é a matriz de probabilidades de transição, `nu0` é o valor inicial para o limite superior da norma H_∞, `tol` é o erro máximo (em termos absolutos) admissível para a solução final, `op1` e `op2` são as opções para a rotina de `hinfsopt` (digite `help hinfsopt` para maiores informações) e `v` controla a saída de informações sobre o andamento do algoritmo misto (`v=0` significa "sem informações").

`Fvec` contém o controlador desejado, `nuopt` o limite superior para a norma H_∞ e `Xvec` a solução para as EARA associadas ao problema H_∞.

Caso o problema seja inadmissível para o valor `nu0` fornecido pelo usuário, a rotina trata de procurar um valor admissível (dobrando sucessivamente `nu0` até que exista solução para o problema), iniciando o procedimento de bissecção a partir desse valor.

IMPORTANTE: Uma escolha adequada das opções (em especial `op2`) é essencial para a convergência.

Algoritmo

`solvhinf` utiliza o algoritmo de `hinfsopt` combinado com o tradicional método da bissecção.

Veja também

`hinfsopt`, `solvhinf`

specrad

Propósito

Cálculo de $r_\sigma(\mathcal{L})$ (raio espectral do operador $\mathcal{L}$).

Sinopse

```
sig = specrad(Gvec,P)
```

Descrição

`Gvec` contém a N-upla de matrizes de transição de estado para o sistema e `P` contém a matriz de probabilidades de transição.

Note que para o sistema ser estável deve-se ter `sig` < 1.

Problemas conhecidos

Quando esta função é compilada em sistemas baseados em MS-Windows e Matlab 4.2 a seguinte mensagem de erro é apresentada na tela:

```
You have chosen a reserved DOS device name for your filename.
Please choose another valid filename.
```

Essa mensagem pode, a princípio, ser ignorada.

Veja também

`cloopvec`, `stack`

stack

Propósito

Compacta N-uplas

Sinopse

```
Xvec = stack(X1,X2,X3,...)
```

Descrição

`stack` faz parte do pacote LMITOOL. Veja a seção 1.3 do guia do usuário ou digite `help stack` para maiores informações.

unstack

Propósito

Descompacta *N*-uplas

Sinopse

```
[X1,X2,X3,...] = unstack(Xvec)
Xn = unstack(Xvec,n)
```

Descrição

`unstack` faz parte do pacote LMITOOL. Veja a seção 1.3 do guia do usuário ou digite `help unstack` para maiores informações.