Eigenvalue Tools

Preliminary Beta version, May 29, 2016

Michiel E. Hochstenbach

Department of Mathematics and Computer Science
TU Eindhoven
PO Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

This collection of MATLAB software contains implementations of several methods for eigenvalue type problems.

Please note:

- **The current toolbox is a Beta version containing beta versions of a limited number of methods.**
- **It is the intention that several new routines are added in the coming weeks. A version 1.0 release is planned for the summer of 2016.**
- **Bugs may be present; feedback and bug reports are greatly appreciated, as quick fixes may be available.**

Although effort is made to make the codes efficient, the routines should be considered as research codes without industrial speed, strength, and stability. The toolbox is distributed under the terms of the MIT Expat License.

Acknowledgments

This work has been supported by an NWO Vidi research grant supported by the Dutch Research Council for Physical Sciences. Some codes reflects joint work with Sarah W. Gaaf, Henk A. van der Vorst, Jos L.M. van Dorsselaer, and Ian N. Zwaan. The layout of this manual was inspired by that of AIR Tools by Per Christian Hansen.

Overview of Eigenvalue Tools

(with page numbers of description)

General comments

- The main idea of the toolbox is the **fast computation** of **useful approximate information** for **large sparse matrices** from **low-dimensional subspaces**.
- Exact LU decompositions are generally avoided; at most, inexact LU decompositions are used. Exceptions are `condprob`, `ilu0`, `krylov_extended`.
- In all procedures, use the empty matrix `[]` to use the default value (if any) for an input variable.
- In the rest of the document, the functions are described alphabetically, not in the order listed above.

The fields for functions that have an `opts` option input variable may include:

Variable	Meaning	Default
`nr`	Number of values	1
`target`	Target $\in \mathbb{R} \cup \{\infty\}$ or $\mathbb{C} \cup \{\infty\}$ (default depends on function)	–
`v1`	Starting vector	`randn1`
`tol`	Tolerance for residual	10^{-6}
`absrel`	Absolute or relative tolerance outer iteration	`'rel'`
	`'abs'` Absolute tolerance	
	`'rel'` Relative tolerance (w.r.t. $\|A\|_1$)	
`maxit`	Max number of restarts	1000
`mindim`	Min dimension of Krylov spaces	10
`maxdim`	Max dimension of Krylov spaces	20
`info`	Info level j	0
	0 No info, recommended in most cases for speed	
	1 Modest information at beginning and end of process	
	2 Full info at beginning, end, and after every iteration	
	≥ 3 Info at beginning, end, and in every jth step only	

Comparison of functions that have some similarities:

`lanprob`	`krylov_schur_sym`
No restarts	Restarts
$\lambda_{\max}$ and rough estimate $\lambda_{\min}$	One or more largest λ's
No eigenvectors	Eigenvectors
Less accurate	More accurate
Faster	Slower

`normprob`	`krylov_schur_svd`
No restarts	Restarts
$\sigma_{\max}$ and rough estimate $\sigma_{\min}$	One or more largest σ's
No singular vectors	Singular vectors
Less accurate	More accurate
Faster	Slower

bisection

Tries to find a zero of a univariate function on a given interval.

Syntax

```
m = bisection(f, a, b)
[m, k, fm, flag, varargout] = bisection(f, a, b)
bisection(f, a, b, maxit, tol, mindx, info)
```

Description

Tries to finds an approximate zero `m` of f on the interval $[a, b]$.
`m` may not be a zero if there is none.

`k` contains the number of performed steps.

`fm` is the function value $f(m)$.

`flag == 0` if the process stopped successfully and `flag == 1` if it did not stop in `maxit` steps.

The function stops if one of the following three conditions has been met:

- $|$`fm`$| <$ `tol`;
- two consecutive approximation differ less than `mindx` relatively;
- `maxit` steps have been performed.

`varargout` is an option to capture other interesting output when an evaluation with f is an expensive operation that also produces important other information.

Apart from the fact that this is an important basic routine, the function is used by `lanprob_delta` in this toolbox.

Example

```
[m, k] = bisection(@(x) cos(x), 1, 2)
```

condprob

Extended Lanczos bidiagonalization with probabilistic upper bound for the (2-norm) condition number of a nonsingular matrix.

Syntax

```
[low, up] = condprob(A, Ainv)
[low, up, k] = condprob(A, Ainv)
condprob(A, Ainv, v1, epsilon, ratio, info)
```

Description

Carries out extended Lanczos bidiagonalization method for the approximation of the 2-norm condition number $\kappa(A) = \|A\|_2 \, \|A^{-1}\|_2$.
`low` is a lower bound for $\kappa(A)$, while `up` is a probabilistic upper bound that holds with probability at least $1 - \varepsilon$.

Although the second argument `Ainv` is not needed (an LU-decomposition will be computed if it is absent), it is recommended to precompute it by

```
Ainv = lu0(A);
```

as this step will generally be the costliest. This routine is one of the few exceptions in the toolbox that needs an exact LU decomposition.

For the probabilistic bounds the vector `v1` should be random on the unit sphere, as generated by the function `randn1`. Passing a `v1` ensures that the experiment can be repeated. The sizes of `A` and `v1` should be compatible.

`epsilon` (default: 0.01) indicates the probability level: the probabilistic upper bound holds with probability $\geq 1 - \varepsilon$.

`ratio` (default: 0.01) indicates the desired upper bound for the ratio `up / low`.

`k` is the number of matrix-vector products with A, A^*, A^{-1}, and A^{-*}.

`info` (default: 0) indicates the desired output: choose 0 for no output, 1 for a small table at the end, and 2 for full output for every iteration step.

Example

```
A = tridiag(0, -1, 1, 10000);
Ainv = lu0(A);
b = randn1(10000);
[low, up, k] = condprob(A, Ainv, b, [], [], 1);
```

See also

`condest`, `normprob`, `lanprob`, `lanprob_delta`, `krylov_ata_extended`.

For more details

S.W. Gaaf, M.E. Hochstenbach
Probabilistic bounds for the matrix condition number with extended Lanczos bidiagonalization
SIAM J. Sci. Comp. 37(5), pp. S581–S601, 2015

element

Element of matrix or function.

Syntax

```
x = element(A, i, j)
x = element(f, i, j, n)
```

Description

Pick element (i, j) from matrix A. This routine is practical to avoid incorrect syntax such as `(A*B)(1,1)`, or when one would like the (1,1) element of a matrix function.

Take `i` empty for the `j`th column, or `j` empty for the `i`th row.

`n` has to be specified only when `f` is a function $f : \mathbb{C}^n \to \mathbb{C}^m$.

Example

```
A = randn(3);
element(A*A, 2, 1)
f = @(x) A*x;
element(f, 2, 1, 3)
```

krylov_ata

Construct orthonormal bases for the Lanczos bidiagonalization subspaces.

Syntax

```
[U, V] = krylov_ata(A, b, k)
[U, V, alpha, beta] = krylov_ata(A, b, k)
krylov_ata(A, b, k, full, reortho)
```

Description

Construct an orthonormal basis V_k for the subspace $\mathcal{K}_k(A^*A, \mathbf{b}) = \text{span}(\mathbf{b}, (A^*A)\mathbf{b}, \ldots, (A^*A)^{k-1}\mathbf{b})$ as well as an orthonormal basis U_k for $\mathcal{K}_k(AA^*, A\mathbf{b})$.
Optionally also the k `alpha`'s which are the diagonal elements, and $k-1$ `beta`'s which are the superdiagonal elements of the bidiagonal projected matrix $B_k = B_{k,k} = U_k^* A V_k$:
`B = diag(alpha) + diag(beta, 1)`.
Moreover, $AV_k = U_k B_{k,k}$.

`A` may be a (square or rectangular) matrix or function such that `A(x)` carries out the matrix-vector product with A. The sizes of `A` and `b` should be compatible.

The output is as described when `full == 0`.
When `full == 1` (default), `V` contains the orthonormal basis of dimension $k+1$ of $\mathcal{K}_{k+1}(A^*A, \mathbf{b})$, `beta` is of length k instead of $k-1$ and $B_{k+1,k}$ is a $k+1 \times k$ matrix. As before $AV_k = U_k B_{k,k}$, but in this case we also have $A^*U_k = V_{k+1}B_{k,k+1}^*$.

When `reortho == 2` (default value), in every step orthogonalization of both V and U with respect to the previous vector is carried out, followed by reorthogonalization on the full basis. When `reortho == 1`, this is done only for the V basis.
If `reortho == 0` (usually not recommended), only one orthogonalization step is carried out for both V and U on the previous vector. This is cheaper but less stable since orthogonality in V may be lost, as, for instance, $\|I - V^*V\|$ may increase during the process.

Example

Approximate some of the singular values of a discretization of the first-order derivative of dimension 10000 from a 10-dimensional subspace.

```
A = tridiag(0, -1, 1, 10000);
b = randn1(10000);
[U, V, alpha, beta] = krylov_ata(A, b, 10, 0);
B = diag(alpha) + diag(beta, 1);
S = svd(B)
```

See also

`normprob`, `krylov_schur_svd`, `krylov_sym`.

krylov_ata_extended

Construct an orthonormal basis for the extended Lanczos bidiagonalization subspace $\text{span}((A^*A)^{-k+1}\mathbf{b}, \ldots, (A^*A)^{-1}\mathbf{b}, \mathbf{b}, (A^*A)\mathbf{b}, \ldots, (A^*A)^{k-1}\mathbf{b})$.

Syntax

```
[U, V, C, D, H, K] = krylov_ata_extended(A, Ainv, b, k)
```

Description

The construction is described in the paper mentioned below. $H_k = U_k^T A V_k$ and $K_k = V_k^T A^{-1} U_k$ are tridiagonal matrices with an additional special structure, as well as each other's inverse. These matrices may be used to approximate the largest and smallest singular value, and therefore the condition number of A.

`A` must be a nonsingular matrix. The sizes of `A` and `b` should be compatible.

Example

Approximate the condition number of a discretization of the first-order derivative of dimension 1000:

```
A = tridiag(0, -1, 1, 1000);
Ainv = lu0(A);
b = randn1(1000);
[U, V, C, D, H, K] = krylov_ata_extended(A, Ainv, b, 4);
smax_low = norm(H)
smin_up = 1 / norm(K)
kappa_low = smax_low / smin_up
kappa = cond(full(A))
```

See also

`condprob`, `krylov_ata_extended_expand`, `krylov_ata`.

For more details

S.W. Gaaf, M.E. Hochstenbach
Probabilistic bounds for the matrix condition number with extended Lanczos bidiagonalization
SIAM J. Sci. Comp. 37(5), pp. S581–S601, 2015

krylov_ata_extended_expand

Expand the orthonormal bases for extended Lanczos bidiagonalization.

Syntax

```
[U, V, C, D, H, K] = ...
      krylov_ata_extended_expand(A, Ainv, U, V, C, D, H, K, m)
```

Description

Expand the orthonormal bases for extended Lanczos bidiagonalization by m.
See `krylov_ata_extended` for a description.

This routine is used by `condprob`.

See also

`condprob`, `krylov_ata_extended`.

krylov_schur_svd

Krylov–Schur method to compute one or more singular values, in particular that of $\|A\|$, and the corresponding singular vectors.

Syntax

```
sigma = krylov_schur_svd(A)
[sigma, V, U, hist, mvs] = krylov_schur_svd(A, opts)
```

Description

Determine some of the largest singular values by a restarted Lanczos bidiagonalization type method. A may be square or rectangular. With $S = \mathrm{diag}(\texttt{sigma})$, we have $AV = US$ (up to a small error) and $\|A^T U - VS\| \leq \texttt{tol}$ (or relatively, see p. 3). V and U are the right and left singular vectors, respectively.

Fields of `opts` are listed on p. 3. Choose `opts.target = Inf` (default) for the largest singular values, and `opts.target = 0` for the smallest. Note that the approximation of the largest singular values is usually not very hard, but that of the smallest singular values is much harder. In many cases, one will need alternative techniques using inexact inverses (such as `jdsvd`) or even exact inverses (such as `condprob`) for this task.

`hist` contains a history of $\|A^T U_j - V_j S_j\|$ during the iterations, with `mvs` the number of matrix-vector products carried out with A **or** A^*, which we can visualize by `semilogy(mvs, hist)`.

`normprob` also computes the largest singular value, but does so without restarts, so that additionally a rough estimate for $\sigma_{\min}(A)$ may be given. `normprob` in addition renders a probabilistic upper bound. By employing restarts, `krylov_schur_svd` focuses on the largest (or possibly the smallest) singular values, and will attempt to compute them to greater accuracy, including the corresponding vectors. See p. 3 for more details.

Examples

Approximate 3 largest the singular triplets of a matrix.

```
A = spdiag(1:10000);
opts.nr = 3;
opts.v1 = randn1(10000);
opts.info = 10;
[S, V, U] = krylov_schur_svd(A, opts);
```

`krylov_schur_svd` should usually be superior over `normest` for the computation of $\|A\|$.

```
A = spdiag(1:10000);
krylov_schur_svd(A)
normest(A)
```

See also

`krylov_ata, normprob`

For more details

M. Stoll
A Krylov–Schur approach to the truncated SVD
Lin. Alg. Appl. 436(8), pp. 2795–2806, 2012.

krylov_schur_sym

Krylov–Schur method to compute one or more of the largest eigenvalues and the corresponding eigenvectors of Hermitian matrices.

Syntax

```
lambda = krylov_schur_sym(A)
[lambda, X, hist, mvs] = krylov_schur_sym(A, opts)
```

Description

Determine some of the largest eigenvalues by a restarted Lanczos type method for a square matrix A. With $D = \mathrm{diag}(\texttt{lambda})$, we have $\|AX - DX\| \leq \texttt{tol}$ (or relatively, see p. 3). X are the (right) eigenvectors.

For the smallest eigenvalues, apply the method to $-A$.

Fields of `opts` are listed on p. 3. Note that the approximation of extremal eigenvalues is usually not very hard, but that of the interior eigenvalues is much harder.

`hist` contains a history of $\|AX_j - X_j D_j\|$ during the iterations, with `mvs` the number of matrix-vector products carried out with A, which we can visualize by `semilogy(mvs, hist)`.

`lanprob` also computes the largest eigenvalue, but does so without restarts, so that additionally a rough estimate for $\lambda_{\min}(A)$ may be given. `lanprob` in addition renders a probabilistic upper bound. By employing restarts, `krylov_schur_sym` focuses on the largest eigenvalues, and will attempt to compute them to greater accuracy, including the corresponding vectors. See p. 3 for more details.

Examples

Approximate the largest eigenpair of a matrix.

```
A = spdiag(1:10000);
opts.v1 = randn1(10000);
[X,D] = krylov_schur_sym(A);
```

See also

`krylov_sym, lanprob`

krylov_sym

Construct orthonormal basis for Krylov subspace $\mathcal{K}_k(A, \mathbf{b})$ for Hermitian A by the Lanczos method.

Syntax

```
V = krylov_sym(A, b, k)
[V, alpha, beta] = krylov_sym(A, b, k)
krylov_sym(A, b, k, full, reortho)
```

Description

Construct an orthonormal basis V_k for the Krylov subspace $\mathcal{K}_k(A, \mathbf{b}) = \text{span}(\mathbf{b}, A\mathbf{b}, \ldots, A^{k-1}\mathbf{b})$ for a Hermitian matrix. Optionally also the k `alpha`'s which are the diagonal elements, and $k-1$ `beta`'s which are the superdiagonal and subdiagonal elements of the Hermitian tridiagonal projected matrix $T_k = T_{k,k} = V_k^* A V_k$:
`T = diag(alpha) + diag(beta, 1) + diag(beta, -1)`.

`A` may be a square matrix or function such that `A(x)` carries out the matrix-vector product with A. The code assumes that it is symmetric or Hermitian, so that 3-term recurrences are obtained. The sizes of `A` and `b` should be compatible.

The output is as described when `full == 0`.
When `full == 1` (default), `V` contains the orthonormal basis of dimension $k+1$ of $\mathcal{K}_{k+1}(A, \mathbf{b})$, `beta` is of length k instead of $k-1$, such that
`T = diag(alpha) + diag(beta(1:end-1), 1) + diag(beta, -1)`
is a $k+1 \times k$-matrix. In this case, $AV_k = V_{k+1}T_{k+1,k}$.

When `reortho == 1` (default value, recommended), in every step orthogonalization with respect to the previous two vectors is carried out, followed by reorthogonalization on the full basis.
If `reortho == 0`, only one orthogonalization step is carried out, and only on the previous two vectors. This is cheaper but less stable since orthogonality in V may be lost, as, for instance, $\|I - V^*V\|$ may increase during the process.

Example

Approximate some of the eigenvalues of a discretization of a 1-D Laplacian of dimension 10000 from a 10-dimensional subspace.

```
A = tridiag(1, -2, 1, 10000);
b = randn1(10000);
[V, alpha, beta] = krylov_sym(A, b, 10, 0);
T = diag(alpha) + diag(beta, 1) + diag(beta, -1);
D = eig(T)
```

See also

`krylov`

lanprob

Lanczos with probabilistic upper bounds for the largest eigenvalue of a real symmetric matrix.

Syntax

```
[low, up] = lanprob(A)
[low, up, lmin, k] = lanprob(A)
lanprob(A, v1, epsilon, ratio, info)
```

Description

Carries out the standard Lanczos method for the approximation of the largest eigenvalue $\lambda_{\max}$. `low` is a lower bound for $\lambda_{\max}$, while `up` is a probabilistic upper bound that holds with probability at least $1 -$ `epsilon`.

For the probabilistic bounds the vector `v1` should be random on the unit sphere, as generated by the function `randn1`. Passing a `v1` ensures that the experiment can be repeated. The sizes of `A` and `v1` should be compatible.

`epsilon` (default: 0.01) indicates the probability level: the probabilistic upper bound holds with probability $\geq 1 - \varepsilon$.

`ratio` (default: 0.01) indicates the desired upper bound for the ratio `(up-low) / (low-lmin)`, where `lmin` is an approximation to the smallest eigenvalue as computed during the process (changing in every iteration). A nice feature of this stopping criterion is that it is invariant under shifts $A - \sigma I$ of the matrix. `lmin` may be a good approximation to $\lambda_{\min}$ in its own right.

`k` is the number of performed steps, and therefore also of matrix-vector products.

`info` (default: 0) indicates the desired output: choose 0 for no output, 1 for a small table at the end, and 2 for full output for every iteration step.

For the smallest eigenvalue of a real symmetric matrix, apply this function to $-A$.

Example

```
A = spdiag(1:10000);
b = randn1(10000);
[low, up, lmin, k] = lanprob(A, b, [], [], 1);
```

See also

`normprob`, `condprob`, `lanprob_delta`, `krylov_sym`.

For more details

J.L.M. van Dorsselaer, M.E. Hochstenbach, H.A. van der Vorst
Computing probabilistic bounds for extreme eigenvalues of symmetric matrices with the Lanczos method
SIAM J. Matrix Anal. Appl. 22(3), pp. 837–852, 2000

lanprob_delta

Computes the "delta" variable for probabilistic methods.

Syntax

```
[delta, k] = lanprob_delta(n, epsilon)
```

Description

For a vector $\mathbf{v} \in \mathbb{R}^n$ that is random on the unit sphere S^{n-1}, this function computes the value δ such that the first (or any other component) satisfies $P(|\mathbf{v}_1| < \delta) = 1 - \varepsilon$.

The importance of this is that a random vector is unlikely to be (almost) orthogonal to the unknown eigenvector of interest, and is therefore a safe choice as an initial vector for iterative methods.

`epsilon` should be a small probability, default 0.01, corresponding with probabilistic bounds that hold with probability at least 99%.

Example Of 10000 components, approximately 100 (1%) should be $< \delta$, corresponding to $\varepsilon = 0.01$.

```
delta = lanprob_delta(10000);
b = randn1(10000);
length(b(abs(b) < delta))
```

See also

`lanprob, normprob, condprob`.

For more details

J.L.M. van Dorsselaer, M.E. Hochstenbach, H.A. van der Vorst
Computing probabilistic bounds for extreme eigenvalues of symmetric matrices with the Lanczos method
SIAM J. Matrix Anal. Appl. 22(3), pp. 837–852, 2000

M.E. Hochstenbach
Probabilistic upper bounds for the matrix two-norm
J. Sci. Comp. 57 (3), pp. 464–476, 2013

lu0

LU decomposition with extras.

Syntax

```
M = lu0(A)
```

Description

For a nonsingular matrix A, this function renders a function `M` with 2 input arguments such that `M(x,0)` performs `A\x` while `M(x,1)` performs `A'\x`. The advantage over `A\x` is the fact that the LU decomposition implicitly present in `M` can be reused for several right-hand sides.

Note that while most procedures of this toolbox only use matrix-vector products, or inexact decompositions at most, this routine computes an exact inverse via an LU decomposition.

This routine uses Tim Davis' `cs_usolve`, `cs_lsolve`, `cs_utsolve`, and `cs_ltsolve`, which are included in the zip-file.

Example

```
A = tridiag(0, -1, 1, 10000);
M = lu0(A);
b = randn1(10000);
x1 = A\b;
x2 = A'\b;
y1 = M(b,0);
y2 = M(b,1);
norm(x1-y1), norm(x2-y2)
```

mv

Performs matrix-vector product.

Syntax

```
y = mv(A, x)
y = mv(A, x, transp_flag)
```

Description

Carries out a matrix-vector product.
The main use of this routine is the handling of functions `A` instead of matrices.

`A` may be a matrix or function.
When `A` is a function, `A(x)` or `A(x,0)` should carry out the matrix-vector product with A, while `A(x,1)` should carry out the matrix-vector product with A^*.
When `A == []`, it is assumed to be the identity.

The sizes of `A` and `x` should be compatible.

If `transp_flag` is absent, or `transp_flag == 0`, multiplication by A is performed.
If `transp_flag == 1`, multiplication by A^* is done.

normprob

Lanczos bidiagonalization with probabilistic upper bound for the 2-norm (= the largest singular value) of a matrix.

Syntax

```
[low, up] = normprob(A)
[low, up, smin, k] = normprob(A)
normprob(A, v1, epsilon, ratio, info)
```

Description

Carries out Lanczos bidiagonalization method for the approximation of the largest singular value $\|A\|_2 = \sigma_{\max}(A)$.
`low` is a lower bound for $\sigma_{\max}$, while `up` is a probabilistic upper bound that holds with probability at least $1 -$ `epsilon`.

For the probabilistic bounds the vector `v1` should be random on the unit sphere, as generated by the function `randn1`. Passing a `v1` ensures that the experiment can be repeated. The sizes of `A` and `v1` should be compatible.

`epsilon` (default: 0.01) indicates the probability level: the probabilistic upper bound holds with probability $\geq 1 - \varepsilon$.

`ratio` (default: 0.01) indicates the desired upper bound for the ratio `up / low`.

The optional output `smin` is an approximation to the smallest singular value $\sigma_{\min}$ as computed during the process. The ratio `low / smin` gives a lower bound for the condition number $\kappa(A) = \sigma_{\max}/\sigma_{\min}$. However, `smin` will usually be a quite poor approximation, as it is computed from a standard Krylov space. If an exact LU decomposition is affordable, the function `condprob` which uses extended Krylov spaces will give a much better approximation to $\kappa(A)$.

`k` is the number of matrix-vector products with A and A^*.

`info` (default: 0) indicates the desired output: choose 0 for no output, 1 for a small table at the end, and 2 for full output for every iteration step.

`krylov_schur_svd` may also compute $\|A\|$; differences being that `normprob` uses no restarts, also computes a probabilistic upper bound, and a rough estimate of $\sigma_{\min}$. `krylov_schur_svd` also may compute the corresponding right and left singular vectors.

Example

```
A = tridiag(0, -1, 1, 10000);
b = randn1(10000);
[low, up, smin, k] = normprob(A, b, [], [], 1);
```

See also

`krylov_schur_svd, lanprob, condprob, lanprob_delta, krylov_ata`.

For more details

M.E. Hochstenbach
Probabilistic upper bounds for the matrix two-norm
J. Sci. Comp. 57 (3), pp. 464–476, 2013

pentadiag

Create a sparse pentadiagonal matrix with given stencil.

Syntax

```
A = pentadiag(a, b, c, d, e, m, n)
```

Description

Create a sparse tridiagonal $m \times n$ matrix with stencil $[a \;\; b \;\; c \;\; d \;\; e]$.

Example

100×100 discretization of fourth-order derivative.

```
L4 = tridiag(1, -4, 6, -4, 1, 100);
L2 = tridiag(1, -2, 1, 100);
L4 - L2*L2
```

See also

`tridiag`, `spdiag`, `spdiags`.

randn1

Random vector on the unit sphere.

Syntax

```
v = randn1(n)
```

Description

Create a random vector on the unit sphere $S^{n-1} = \{ \mathbf{x} : \|\mathbf{x}\| = 1 \} \subset \mathbb{R}^n$. The joint probability density function of `v = randn(n,1)` is, up to a constant, equal to $e^{-\|\mathbf{v}\|}$, so only depends on the distance to the origin. Division by its norm ensures randomness on the unit sphere.

This choice of vector is used by the methods that generate probabilistic upper bounds.

Example

This angle plot shows approximately a straight line.

```
for j = 1:1000
    V(:,j) = randn1(2);
end
phi = sort(atan2(V(2,:), V(1,:)));
plot(phi)
```

See also

`randn`.

rgs

Repeated (classical) Gram–Schmidt.

Syntax

```
[y, h] = rgs(x, U)
y = rgs(x, U)
```

Description

Carries out repeated Gram–Schmidt to compute $\mathbf{y} = (I - UU^*)\mathbf{x}$ for an $n \times k$ orthonormal basis U, where $k < n$ and usually $k \ll n$. The coefficients $U^*\mathbf{x}$ are captured in `h`.

The orthogonal projection $I - UU^*$ is repeated when $\mathbf{y} = (I - UU^*)\mathbf{x}$ is of small norm, since in this case the rounding errors may render an inaccurate vector $\mathbf{y}$. Although mathematically $(I - UU^*)^2 = I - UU^*$, the difference in finite precision arithmetic may be considerable. The projection is performed at most twice ("twice is enough").

This routine is used in Krylov methods involving non-Hermitian matrices, where the `h` vectors may appear in a Hessenberg matrix.

Example

```
x = randn1(100);
y = x + 1e-12 * randn1(100);
z1 = y - x*(x'*y);            % Gram--Schmidt
z1 = z1 / norm(z1);
z2 = rgs(y, x);               % Repeated Gram--Schmidt
abs(z1'*x)                    % Not small
abs(z2'*x)                    % Small
```

spdiag

Create a sparse diagonal matrix with given entries.

Syntax

```
A = spdiag(v)
```

Description

Create a sparse diagonal matrix of size `length(v)` with the entries of the vector `v` on the diagonal.

Example

```
A = spdiag(1:10000);
```

See also

`spdiags`, `tridiag`.

tridiag

Create a sparse tridiagonal matrix with given stencil.

Syntax

```
A = tridiag(a, b, c, m, n)
```

Description

Create a sparse tridiagonal $m \times n$ matrix with stencil $[a \;\; b \;\; c]$.

Examples

100×100 discretization of 1-D Laplacian.

```
A = tridiag(1, -2, 1, 100);
```

99×100 discretization of first-order derivative.

```
L = tridiag(1, -2, 1, 99, 100);
L(1,1) = -1;
norm(L*ones(100,1))
```

See also

`spdiag`, `spdiags`.

unv

Standard basis vector (of unit length).

Syntax

```
x = unv(j, n)
x = unv(j, n, alpha)
```

Description

Create standard basis vector $\mathbf{e}_j \in \mathbb{R}^n$ or $\mathbb{C}^n$; all elements zero except for a 1 at the jth position. With a third element `alpha`, $\mathbf{x}$ has α at the jth position.

The `j` argument may also be a vector of $k < n$ positive integers $\in \{1, \ldots, n\}$, in which case $\mathbf{x}$ will be $n \times k$ containing k standard basis vectors.