PyLinearSolver

Release 0.1.1

Aug 03, 2020

Contents:

CHAPTER 1

About the Package

PyLinearSolver is a python iterface for many well-known linear solvers packages written in different languages like C, C++, julia, etc. . . The package uses the existing implementation from the original source and build a wrapper layer for a python development.

NOTE All rights are preserved to the authers and developers of the original packages.

CHAPTER 2

Installation

2.1 Dependencies

To work with PyLinearSolver packages, you need to install the dependencies for them.

Iterative Solvers

- Install Julia
- It is preferable to install a custom python version using pyenv due to an issue with PyJulia.

2.2 PyLinearSolver

Simply you need to call the following command:

```
pip install PyLinearSolver
```

NOTE In case of working with Iterative Solvers, you need to install the dependencies for PyJulia through the following:

```
import julia
julia.install()
```

CHAPTER 3

PyLinearSolver packages

3.1 IterativeSolvers

`PyLinearSolver.IterativeSolvers.`**`bicgstabl`**(*A*, *b*, *l*, ***kwargs*)

The function is a wrapper for the julia implementation of BiCGStab(l) solver in IterativeSolver.jl package. BiCGStab(l) solves the problem $Ax = b$ approximately for x where A is a general, linear operator and b the right-hand side vector. The methods combines BiCG with l GMRES iterations, resulting in a short-reccurence iteration. As a result the memory is fixed as well as the computational costs per iteration.

Arguments

- A: linear operator;
- b: right hand side (vector);
- l::Int = 2: Number of GMRES steps.

Keywords

- max_mv_products::Int = size(A, 2): maximum number of matrix vector products.

For BiCGStab(l) this is a less dubious term than "number of iterations";

- Pl = Identity(): left preconditioner of the method;
- tol::Real = sqrt(eps(real(eltype(b)))): tolerance for stopping condition $|r_k|/|r_0|tol$. Note that (1) the true residual norm is never computed during the iterations, only an approximation; and (2) if a preconditioner is given, the stopping condition is based on the preconditioned residual.

Output

if log is false

- x: approximate solution.

if log is true

- x: approximate solution;
- history: convergence history.

`PyLinearSolver.IterativeSolvers.`**`cg`**(*A*, *b*, ***kwargs*)

The function is a wrapper for the julia implementation of Conjugate Gradients solver in IterativeSolver.jl package. Conjugate Gradients solves $Ax = b$ approximately for x where A is a symmetric, positive-definite linear operator and b the right-hand side vector. The method uses short recurrences and therefore has fixed memory costs and fixed computational costs per iteration.

Arguments

- A: linear operator
- b: right-hand side.

Keywords

- statevars::CGStateVariables: Has 3 arrays similar to x to hold intermediate results.
- initially_zero::Bool: If true assumes that iszero(x) so that one matrix-vector product can be saved when computing the initial residual vector
- Pl = Identity(): left preconditioner of the method. Should be symmetric, positive-definite like A
- tol::Real = sqrt(eps(real(eltype(b)))): tolerance for stopping condition $|r_k|/|r_0|tol$
- maxiter::Int = size(A,2): maximum number of iterations
- verbose::Bool = false: print method information
- log::Bool = false: keep track of the residual norm in each iteration.

Output

if log is false

- x: approximated solution.

if log is true

- x: approximated solution.
- ch: convergence history.

ConvergenceHistory keys

- tol => ::Real: stopping tolerance.
- resnom => ::Vector: residual norm at each iteration.

`PyLinearSolver.IterativeSolvers.`**`chebyshev`**(*A*, *b*, *lamda_min*, *lamda_max*, ***kwargs*)

The function is a wrapper for the julia implementation of Chebyshev Iteration solver in IterativeSolver.jl package. Chebyshev iteration solves the problem $Ax = b$ approximately for x where A is a symmetric, definite linear operator and b the right-hand side vector. The methods assumes the interval $[min, max]$ containing all eigenvalues of A is known, so that x can be iteratively constructed via a Chebyshev polynomial with zeros in this interval. This polynomial ultimately acts as a filter that removes components in the direction of the eigenvectors from the initial residual. The main advantage with respect to Conjugate Gradients is that BLAS1 operations such as inner products are avoided.

Arguments

- A: linear operator;
- b: right-hand side;
- λmin::Real: lower bound for the real eigenvalues
- λmax::Real: upper bound for the real eigenvalues

Keywords

- initially_zero::Bool = false: if true assumes that iszero(x) so that one matrix-vector product can be saved when computing the initial residual vector;
- tol: tolerance for stopping condition $|r_k|/|r_0|tol$.
- maxiter::Int = size(A, 2): maximum number of inner iterations of GMRES;
- Pl = Identity(): left preconditioner;
- log::Bool = false: keep track of the residual norm in each iteration;
- verbose::Bool = false: print convergence information during the iterations.

Output

if log is false

- x: approximate solution.

if log is true

- x: approximate solution;
- history: convergence history.

`PyLinearSolver.IterativeSolvers.`**`gauss_seidel`**(*A*, *b*, ***kwargs*)

The function is a wrapper for the julia implementation of Gauss-Seidel solver in IterativeSolver.jl package.

Keywords

- maxiter::Int = 10: maximum number of iterations.

`PyLinearSolver.IterativeSolvers.`**`gmres`**(*A*, *b*, ***kwargs*)

The function is a wrapper for the julia implementation of Restarted GMRES solver in IterativeSolver.jl package. GMRES solves the problem $Ax = b$ approximately for x where A is a general, linear operator and b the right-hand side vector. The method is optimal in the sense that it selects the solution with minimal residual from a Krylov subspace, but the price of optimality is increasing storage and computation effort per iteration. Restarts are necessary to fix these costs.

Arguments

- A: linear operator;
- b: right-hand side.

Keywords

- initially_zero::Bool: If true assumes that iszero(x) so that one matrix-vector product can be saved when computing the initial residual vector;
- tol: relative tolerance;
- restart::Int = min(20, size(A, 2)): restarts GMRES after specified number of iterations;
- maxiter::Int = size(A, 2): maximum number of inner iterations of GMRES;
- Pl: left preconditioner;
- Pr: right preconditioner;
- log::Bool: keep track of the residual norm in each iteration;
- verbose::Bool: print convergence information during the iterations.

Output

if log is false

- x: approximate solution.

if log is true

- x: approximate solution;
- history: convergence history.

`PyLinearSolver.IterativeSolvers.`**`idrs`**(*A*, *b*, *s=8*, ***kwargs*)

The function is a wrapper for the julia implementation of IDR(s) solver in IterativeSolver.jl package. The Induced Dimension Reduction method is a family of simple and fast Krylov subspace algorithms for solving large nonsymmetric linear systems. The idea behind the IDR(s) variant is to generate residuals that are in the nested subspaces of shrinking dimensions.

Arguments

- A: linear operator;
- b: right-hand side.

Keywords

- s::Integer = 8: dimension of the shadow space;
- tol: relative tolerance;
- maxiter::Int = size(A, 2): maximum number of iterations;
- log::Bool: keep track of the residual norm in each iteration;
- verbose::Bool: print convergence information during the iterations.

Output

if log is false

- x: approximate solution.

if log is true

- x: approximate solution;
- history: convergence history.

`PyLinearSolver.IterativeSolvers.`**`jacobi`**(*A*, *b*, ***kwargs*)

The function is a wrapper for the julia implementation of Jacobi solver in IterativeSolver.jl package.

Keywords

- maxiter::Int = 10: maximum number of iterations.

`PyLinearSolver.IterativeSolvers.`**`lsmr`**(*A*, *b*, ***kwrags*)

The function is a wrapper for the julia implementation of Least-squares minimal residual solver in Iterative-Solver.jl package. Minimizes $Axb^2 + x^2$ in the Euclidean norm. If multiple solutions exists the minimum norm solution is returned. The method is based on the Golub-Kahan bidiagonalization process. It is algebraically equivalent to applying MINRES to the normal equations $(AA +^2 I)x = Ab$, but has better numerical properties, especially if A is ill-conditioned.

Arguments

- A: linear operator.
- b: right-hand side.

Keywords

- λ::Number = 0: lambda.

- atol::Number = 1e-6, btol::Number = 1e-6: stopping tolerances. If both are 1.0e-9 (say), the final residual norm should be accurate to about 9 digits. (The final x will usually have fewer correct digits, depending on cond(A) and the size of damp).
- conlim::Number = 1e8: stopping tolerance. lsmr terminates if an estimate of cond(A) exceeds conlim. For compatible systems Ax = b, conlim could be as large as 1.0e+12 (say). For least-squares problems, conlim should be less than 1.0e+8. Maximum precision can be obtained by setting
- atol = btol = conlim = zero, but the number of iterations may then be excessive.
- maxiter::Int = maximum(size(A)): maximum number of iterations.
- log::Bool: keep track of the residual norm in each iteration;
- verbose::Bool: print convergence information during the iterations.

Output

if log is false

- x: approximated solution.

if log is true

- x: approximated solution.
- ch: convergence history.

ConvergenceHistory keys

- atol => ::Real: atol stopping tolerance.
- btol => ::Real: btol stopping tolerance.
- ctol => ::Real: ctol stopping tolerance.
- anorm => ::Real: anorm.
- rnorm => ::Real: rnorm.
- cnorm => ::Real: cnorm.
- resnom => ::Vector: residual norm at each iteration.

`PyLinearSolver.IterativeSolvers.`**`lsqr`**(*A*, *b*, ***kwrags*)

The function is a wrapper for the julia implementation of LSQR solver in IterativeSolver.jl package. Minimizes $Axb^2 + dampx^2$ in the Euclidean norm. If multiple solutions exists returns the minimal norm solution. The method is based on the Golub-Kahan bidiagonalization process. It is algebraically equivalent to applying CG to the normal equations $(AA +^2 I)x = Ab$ but has better numerical properties, especially if A is ill-conditioned.

Arguments

- A: linear operator;
- b: right-hand side.

Keywords

- damp::Number = 0: damping parameter.
- atol::Number = 1e-6, btol::Number = 1e-6: stopping tolerances. If both are 1.0e-9 (say), the final residual norm should be accurate to about 9 digits. (The final x will usually have fewer correct digits, depending on cond(A) and the size of damp).
- conlim::Number = 1e8: stopping tolerance. lsmr terminates if an estimate of cond(A) exceeds conlim. For compatible systems $Ax = b$, conlim could be as large as 1.0e+12 (say). For least-squares problems,

conlim should be less than 1.0e+8. Maximum precision can be obtained by setting atol = btol = conlim = zero, but the number of iterations may then be excessive.

- maxiter::Int = maximum(size(A)): maximum number of iterations.
- verbose::Bool = false: print method information.
- log::Bool = false: output an extra element of type ConvergenceHistory containing extra information of the method execution.

Output

if log is false

- x: approximated solution.

if log is true

- x: approximated solution.
- ch: convergence history.

ConvergenceHistory keys

- atol => ::Real: atol stopping tolerance.
- btol => ::Real: btol stopping tolerance.
- ctol => ::Real: ctol stopping tolerance.
- anorm => ::Real: anorm.
- rnorm => ::Real: rnorm.
- cnorm => ::Real: cnorm.
- resnom => ::Vector: residual norm at each iteration.

`PyLinearSolver.IterativeSolvers.`**`minres`**(*A*, *b*, ***kwargs*)

The function is a wrapper for the julia implementation of MINRES solver in IterativeSolver.jl package. MINRES is a short-recurrence version of GMRES for solving $Ax = b$ approximately for x where A is a symmetric, Hermitian, skew-symmetric or skew-Hermitian linear operator and b the right-hand side vector.

Arguments

- A: linear operator.
- b: right-hand side.

Keywords

- initially_zero::Bool = false: if true assumes that iszero(x) so that one matrix-vector product can be saved when computing the initial residual vector;
- skew_hermitian::Bool = false: if true assumes that A is skew-symmetric or skew-Hermitian;
- tol: tolerance for stopping condition $|r_k|/|r_0|tol$. Note that the residual is computed only approximately;
- maxiter::Int = size(A, 2): maximum number of iterations;
- Pl: left preconditioner;
- Pr: right preconditioner;
- log::Bool = false: keep track of the residual norm in each iteration;
- verbose::Bool = false: print convergence information during the iterations.

Output

if log is false

- x: approximate solution.

if log is true

- x: approximate solution;
- history: convergence history.

`PyLinearSolver.IterativeSolvers.`**`sor`**(*A*, *b*, *w*, ***kwargs*)

The function is a wrapper for the julia implementation of Successive over-relaxation (SOR) solver in Iterative-Solver.jl package.

Arguments

- w: relaxation parameter

Keywords

- maxiter::Int = 10: maximum number of iterations.

`PyLinearSolver.IterativeSolvers.`**`ssor`**(*A*, *b*, *w*, ***kwargs*)

The function is a wrapper for the julia implementation of Symmetric successive over-relaxation (SSOR) solver in IterativeSolver.jl package.

Arguments

- w: relaxation parameter

Keywords

- maxiter::Int = 10: maximum number of iterations.

CHAPTER 4

Examples

4.1 Iterative Solvers

An example for Iterative Solver Wrapper package in PyLinearSolver:

```
from PyLinearSolver import IterativeSolvers
import numpy as np

n=10
A = np.random.rand(n,n)
b= np.random.rand(n)
x= np.zeros(n)
A = A + A.T +2*n*np.identity(n)

x, ch=IterativeSolvers.cg(A,b,verbose=True, log=True)
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index