dolfin_navier_scipy Documentation

Release 1.1.0

highlando

May 26, 2023

CONTENTS

The package *dolfin_navier_scipy (dns)* provides an interface between *scipy* and *FEniCS* in view of solving Navier-Stokes Equations. *FEniCS* is used to perform a Finite Element discretization of the equations. The assembled coefficients are exported as sparse matrices for use in *scipy*. Nonlinear and time-dependent parts are evaluated and assembled on demand. Visualization is done via the *FEniCS* interface to *paraview*.

Contents:

CHAPTER

ONE

CODE REFERENCE

1.1 dolfin_to_sparrays

1.2 stokes_navier_utils

stokes_navier_utils.**get_pfromv**(*v=None*, *V=None*, *M=None*, *A=None*, *J=None*, *fv=None*, *fp=None*, *decouplevp=False*, *solve_M=None*, *symmetric=False*, *cgtol=1e-08*, *stokes_flow=False*, *diribcs=None*, *dbcinds=None*, *dbcvals=None*, *invinds=None*, ***kwargs*)

for a velocity *v*, get the corresponding *p*

Notes

Formula is only valid for constant rhs in the continuity equation

stokes_navier_utils.**get_v_conv_conts**(*vvec=None*, *V=None*, *invinds=None*, *dbcvals=[]*, *dbcinds=[]*, *semi_explicit=False*, *Picard=False*, *retparts=False*)

get and condense the linearized convection

to be used in a Newton scheme

$$(u \cdot \nabla)u \to (u_0 \cdot \nabla)u + (u \cdot \nabla)u_0 - (u_0 \cdot \nabla)u_0$$

or in a Picard scheme

$$(u \cdot \nabla)u \to (u_0 \cdot \nabla)u$$

Parameters

vvec [(N,1) ndarray] convection velocity

V [dolfin.VectorFunctionSpace] FEM space of the velocity

invinds [(N,) ndarray or list] indices of the inner nodes

Picard [Boolean] whether Picard linearization is applied, defaults to *False*

semi_explicit: Boolean, optional whether to return minus the convection vector, and zero convmats as needed for semi-explicit integration, defaults to *False*

retparts [Boolean, optional] whether to return both components of the matrices and contributions to the rhs through the boundary conditions, defaults to *False*

Returns

convc_mat [(N,N) sparse matrix] representing the linearized convection at the inner nodes

rhs_con [(N,1) array] representing $(u_0 \cdot \nabla)u_0$ at the inner nodes

rhsv_conbc [(N,1) ndarray] representing the boundary conditions

stokes_navier_utils.**solve_nse**(*A=None*, *M=None*, *J=None*, *JT=None*, *fv=None*, *fp=None*, *fvtd=None*, *fvss=0.0*, *fvtvd=None*, *fv_tmdp=None*, *iniv=None*, *inip=None*, *lin_vel_point=None*, *stokes_flow=False*, *trange=None*, *t0=None*, *tE=None*, *Nts=None*, *time_int_scheme='cnab'*, *V=None*, *Q=None*, *invinds=None*, *diribcs=None*, *dbcinds=None*, *dbcvals=None*, *diricontbcinds=None*, *diricontbcvals=None*, *diricontfuncs=None*, *diricontfuncmems=None*, *N=None*, *nu=None*, *ppin=None*, *closed_loop=False*, *static_feedback=False*, *dynamic_feedback=False*, *dyn_fb_dict={}*, *dyn_fb_disc='trapezoidal'*, *b_mat=None*, *cv_mat=None*, *vp_output=False*, *vp_out_fun=None*, *vp_output_dict=None*, *vel_nwtn_stps=20*, *vel_nwtn_tol=5e-15*, *nsects=1*, *loc_nwtn_tol=5e-15*, *loc_pcrd_stps=True*, *addfullsweep=False*, *vel_pcrd_stps=4*, *krylov=None*, *krpslvprms={}*, *krplsprms={}*, *clearprvdata=False*, *get_datastring=None*, *data_prfx=''*, *paraviewoutput=False*, *plttrange=None*, *prvoutpnts=None*, *vfileprfx=''*, *pfileprfx=''*, *return_dictofvelstrs=False*, *return_dictofpstrs=False*, *dictkeysstr=False*, *dictkeyformat='.5f'*, *treat_nonl_explicit=True*, *no_data_caching=True*, *treat_nonl_explct=False*, *use_custom_nonlinearity=False*, *custom_nonlinear_vel_function=None*, *datatrange=None*, *dataoutpnts=None*, *return_final_vp=False*, *return_as_list=False*, *return_vp_dict=False*, *return_y_list=False*, *check_ff=False*, *check_ff_maxv=100000000.0*, *verbose=True*, *start_ssstokes=False*, ***kw*)

solution of the time-dependent nonlinear Navier-Stokes equation

$$M\dot{v} + Av + N(v)v + J^T p = f_v$$
$$Jv = f_p$$

using a Newton scheme in function space, i.e. given v_k, we solve for the update like

$$M\dot{v} + Av + N(v_k)v + N(v)v_k + J^T p = N(v_k)v_k + f,$$

and trapezoidal rule in time. To solve an *Oseen* system (linearization about a steady state) or a *Stokes* system, set the number of Newton steps to one and provide a linearization point and an initial value.

Parameters

lin_vel_point [dictionary, optional] contains the linearization point for the first Newton iteration

- Steady State: {{*None*: 'path_to_nparray'}, {'None': nparray}}
- Newton: {*t*: 'path_to_nparray'}

defaults to *None*

dictkeysstr [boolean, optional] whether the *keys* of the result dictionaries are strings instead of floats, defaults to *False*

fvtd [callable f(t), optional] time dependent right hand side in momentum equation

fvtvd [callable f(t, v), optional] time and velocity dependent right hand side in momentum equation

fvss [array, optional] right hand side in momentum for steady state computation

fv_tmdp [callable f(t, v, dict), optional] XXX: this is deprecated

dbcinds: list, optional indices of the Dirichlet boundary conditions

dbcvals: list, optional values of the Dirichlet boundary conditions (as listed in *dbcinds*)

diricontbcinds: list, optional list of dirichlet indices that are to be controlled

diricontbcvals: list, optional list of the vals corresponding to *diricontbcinds*

diricontfuncs: list, optional list like *[ufunc]* where *ufunc: (t, v) -> u* where *u* is used to scale the corresponding *diricontbcvals*

check_ff: boolean, optional whether to check for failures and to invoke a premature break, defaults to *False*

check_ff_maxv: float, optional threshold for norm of velocity that indicates a failure, defaults to *1e8*

krylov [{None, 'gmres'}, optional] whether or not to use an iterative solver, defaults to *None*

krpslvprms [dictionary, optional] v specify parameters of the linear solver for use in Krypy, e.g.,

- initial guess
- tolerance
- number of iterations

defaults to *None*

krplsprms [dictionary, optional] parameters to define the linear system like

- preconditioner

ppin [{int, None}, optional] which dof of *p* is used to pin the pressure, defaults to *None*

stokes_flow [boolean, optional] whether to consider the Stokes linearization, defaults to *False*

start_ssstokes [boolean, optional] for your convenience, compute and use the steady state stokes solution as initial value, defaults to *False*

treat_nonl_explicit= string, optional whether to treat the nonlinearity explicitly, defaults to *False*

use_custom_nonlinearity: boolean, optional whether to use a custom nonlinear velocity function to replace the convection part, defaults to *False*

custom_nonlinear_vel_function: callable f(v), optional the custom nonlinear function (cp. *use_custom_nonlinearity*), defaults to *None*

nsects: int, optional in how many segments the trange is split up. (The newton iteration will be confined to the segments and, probably, converge faster than the global iteration), defaults to *1*

loc_nwtn_tol: float, optional tolerance for the newton iteration on the segments, defaults to *1e-15*

loc_pcrd_stps: boolean, optional whether to init with *vel_pcrd_stps* Picard steps on every section, if *False*, Picard iterations are performed only on the first section, defaults to *True*

addfullsweep: boolean, optional whether to compute the newton iteration on the full *trange*, useful to check and for the plots, defaults to *False*

cv_mat: (Ny, Nv) sparse array, optional output matrix for velocity outputs, needed, e.g., for output dependent feedback control, defaults to *None*

dynamic_feedback: boolean whether to apply dynamic feedback, defaults to *False*

dyn_fb_dict, dictionary

that defines the dynamic observer via the keys

- *ha* observer dynamic matrix
- *hb* observer input matrix
- *hc* observer output matrix
- *drift* observer drift term, e.g., for nonzero setpoints

dyn_fb_disc: string

how to discretize the dynamic observer/controller

- *AB2* explicit scheme (Adams Bashforth)
- *trapezoidal* implicit scheme (Trapezoidal)

Returns

dictofvelstrs [dictionary, on demand] dictionary with time *t* as keys and path to velocity files as values

dictofpstrs [dictionary, on demand] dictionary with time *t* as keys and path to pressure files as values

vellist [list, on demand] list of the velocity solutions, deprecated use *ylist* with *C=I*

stokes_navier_utils.**solve_steadystate_nse**(*A=None*, *J=None*, *JT=None*, *M=None*, *fv=None*, *fp=None*, *V=None*, *Q=None*, *invinds=None*, *diribcs=None*, *dbcvals=None*, *dbcinds=None*, *diricontbcinds=None*, *diricontbcvals=None*, *diricontfuncs=None*, *diricontfuncmems=None*, *return_vp=False*, *ppin=None*, *return_nwtnupd_norms=False*, *N=None*, *nu=None*, *only_stokes=False*, *vel_pcrd_stps=10*, *vel_pcrd_tol=0.0001*, *vel_nwtn_stps=20*, *vel_nwtn_tol=5e-15*, *clearprvdata=False*, *useolddata=False*, *vel_start_nwtn=None*, *get_datastring=None*, *data_prfx=''*, *paraviewoutput=False*, *save_data=False*, *vfileprfx=''*, *pfileprfx=''*, *verbose=True*, ***kw*)

Solution of the steady state nonlinear NSE Problem

using Newton's scheme. If no starting value is provided, the iteration is started with the steady state Stokes solution.

Parameters

A [(N,N) sparse matrix] stiffness matrix aka discrete Laplacian, note the sign!

M [(N,N) sparse matrix] mass matrix

J [(M,N) sparse matrix] discrete divergence operator

JT [(N,M) sparse matrix, optional] discrete gradient operator, set to J.T if not provided

fv, fp [(N,1), (M,1) ndarrays] right hand sides restricted via removing the boundary nodes in the momentum and the pressure freedom in the continuity equation

ppin [{int, None}, optional] which dof of *p* is used to pin the pressure, defaults to *None*

dbcinds: list, optional indices of the Dirichlet boundary conditions

dbcvals: list, optional values of the Dirichlet boundary conditions (as listed in *dbcinds*)

diricontbcinds: list, optional list of dirichlet indices that are to be controlled

diricontbcvals: list, optional list of the vals corresponding to *diricontbcinds*

diricontfuncs: list, optional list like *[ufunc]* where *ufunc: (t, v)* -> *u* where *u* is used to scale the corresponding *diricontbcvals*

return_vp [boolean, optional] whether to return also the pressure, defaults to *False*

vel_pcrd_stps [int, optional] Number of Picard iterations when computing a starting value for the Newton scheme, cf. Elman, Silvester, Wathen: *FEM and fast iterative solvers*, 2005, defaults to *100*

only_stokes: boolean, optional if only compute the stokes solution, defaults to *False*

vel_pcrd_tol [real, optional] tolerance for the size of the Picard update, defaults to *1e-4*

vel_nwtn_stps [int, optional] Number of Newton iterations, defaults to *20*

vel_nwtn_tol [real, optional] tolerance for the size of the Newton update, defaults to *5e-15*

Returns:

—

vel_k [(N, 1) ndarray] the velocity vector, if not *return_vp*, else

(v, p) [tuple] of the velocity and the pressure vector

norm_nwtnupd_list [list, on demand] list of the newton upd errors

1.3 problem_setups

`problem_setups.`**`cyl3D_fems`**(*refinement_level=2*, *scheme='TH'*, *strtobcsobs=''*, *strtomeshfile=''*, *strtophysicalregions=''*, *bccontrol=False*, *verbose=False*)

dictionary for the fem items for the 3D cylinder wake

which is

- the 2D setup extruded in z-direction
- with symmetry BCs at the z-walls

Parameters

scheme [{None, 'CR', 'TH'}] the finite element scheme to be applied, 'CR' for Crouzieux-Raviart, 'TH' for Taylor-Hood, overrides *pdgree*, *vdgree*, defaults to *None*

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to *False*

Returns

femp [a dictionary with the keys:]

- *V*: FEM space of the velocity

- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *dirip*: list of the (Dirichlet) boundary conditions for the pressure
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation
- *charlen*: characteristic length of the setup
- *odcoo*: dictionary with the coordinates of the domain of observation
- *cdcoo*: dictionary with the coordinates of the domain of control
- *uspacedep*: int that specifies in what spatial direction Bu changes. The remaining is constant
- *bcsubdoms*: list of subdomains that define the segments where the boundary control is applied

Notes

parts of the code were taken from the NSbench collection https://launchpad.net/nsbench

__author__ = "Kristian Valen-Sendstad <kvs@simula.no>"
__date__ = "2009-10-01"
__copyright__ = "Copyright (C) 2009-2010 " + __author__
__license__ = "GNU GPL version 3 or any later version"

problem_setups.**cyl_fems**(*refinement_level=2*, *vdgree=2*, *pdgree=1*, *scheme=None*, *inflowvel=1.0*, *bccontrol=False*, *verbose=False*)

dictionary for the fem items for the cylinder wake

Parameters

vdgree [polynomial degree of the velocity basis functions,] defaults to 2

pdgree [polynomial degree of the pressure basis functions,] defaults to 1

scheme [{None, 'CR', 'TH'}] the finite element scheme to be applied, 'CR' for Crouzieux-Raviart, 'TH' for Taylor-Hood, overrides *pdgree*, *vdgree*, defaults to *None*

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to *False*

Returns

femp [a dictionary with the keys:]

- *V*: FEM space of the velocity
- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *dirip*: list of the (Dirichlet) boundary conditions for the pressure
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation

- *charlen*: characteristic length of the setup
- *odcoo*: dictionary with the coordinates of the domain of observation
- *cdcoo*: dictionary with the coordinates of the domain of control
- *uspacedep*: int that specifies in what spatial direction Bu changes. The remaining is constant
- *bcsubdoms*: list of subdomains that define the segments where the boundary control is applied

Notes

TODO: *inflowvel* as input is there for consistency but not processed

parts of the code were taken from the NSbench collection https://launchpad.net/nsbench

__author__ = "Kristian Valen-Sendstad <kvs@simula.no>"
__date__ = "2009-10-01"
__copyright__ = "Copyright (C) 2009-2010 " + __author__
__license__ = "GNU GPL version 3 or any later version"

problem_setups.**drivcav_fems**(*N=10*, *vdgree=2*, *pdgree=1*, *scheme=None*, *bccontrol=None*)
dictionary for the fem items of the (unit) driven cavity

Parameters

N [int] mesh parameter for the unitsquare (N gives 2*N*N triangles)

vdgree [int, optional] polynomial degree of the velocity basis functions, defaults to 2

pdgree [int, optional] polynomial degree of the pressure basis functions, defaults to 1

scheme [{None, 'CR', 'TH'}] the finite element scheme to be applied, 'CR' for Crouzieux-Raviart, 'TH' for Taylor-Hood, overrides *pdgree*, *vdgree*, defaults to *None*

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to false. TODO: not implemented yet but we need it here for consistency

Returns

femp [a dict]

of problem FEM description with the keys:

- *V*: FEM space of the velocity
- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation
- *charlen*: characteristic length of the setup
- *odcoo*: dictionary with the coordinates of the domain of observation
- *cdcoo*: dictionary with the coordinates of the domain of control

problem_setups.**gen_bccont_fems**(*scheme='TH'*, *bccontrol=True*, *verbose=False*, *strtomeshfile=''*, *strtophysicalregions=''*, *inflowvel=1.0*, *inflowprofile='parabola'*, *movingwallcntrl=False*, *strtobcsobs=''*)

dictionary for the fem items for a general 2D flow setup

with

- inflow/outflow
- boundary control

Parameters

scheme [{None, 'CR', 'TH'}] the finite element scheme to be applied, 'CR' for Crouzieux-Raviart, 'TH' for Taylor-Hood, overrides *pdgree*, *vdgree*, defaults to *None*

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to *True*

movingwallcntrl [boolean, optional] whether control is via moving boundaries

Returns

femp [a dictionary with the keys:]

- *V*: FEM space of the velocity
- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *dbcsinds*: list vortex indices with (Dirichlet) boundary conditions
- *dbcsvals*: list of values of the (Dirichlet) boundary conditions
- *dirip*: list of the (Dirichlet) boundary conditions for the pressure
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation
- *charlen*: characteristic length of the setup
- *odcoo*: dictionary with the coordinates of the domain of observation

problem_setups.**gen_bccont_fems_3D**(*scheme='TH'*, *bccontrol=True*, *verbose=False*, *strtomeshfile=''*, *strtophysicalregions=''*, *inflowvel=1.0*, *inflowprofile='parabola'*, *movingwallcntrl=False*, *strtobcsobs=''*)

dictionary for the fem items for a general 3D flow setup

with

- inflow/outflow
- boundary control

Parameters

scheme [{None, 'CR', 'TH'}] the finite element scheme to be applied, 'CR' for Crouzieux-Raviart, 'TH' for Taylor-Hood, overrides *pdgree*, *vdgree*, defaults to *None*

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to *True*

movingwallcntrl [boolean, optional] whether control is via moving boundaries

Returns

femp [a dictionary with the keys:]

- *V*: FEM space of the velocity
- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *dbcsinds*: list vortex indices with (Dirichlet) boundary conditions
- *dbcsvals*: list of values of the (Dirichlet) boundary conditions
- *dirip*: list of the (Dirichlet) boundary conditions for the pressure
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation
- *charlen*: characteristic length of the setup
- *odcoo*: dictionary with the coordinates of the domain of observation

problem_setups.**get_sysmats**(*problem='gen_bccont'*, *scheme=None*, *ppin=None*, *Re=None*, *nu=None*, *charvel=1.0*, *gradvsymmtrc=True*, *bccontrol=False*, *mergerhs=False*, *onlymesh=False*, *meshparams={}*)

retrieve the system matrices for stokes flow

Parameters

problem [{'drivencavity', 'cylinderwake', 'gen_bccont', 'cylinder_rot'}] problem class

N [int] mesh parameter

nu [real, optional] kinematic viscosity, is set to *L/Re* if *Re* is provided

Re [real, optional] Reynoldsnumber, is set to *L/nu* if *nu* is provided

bccontrol [boolean, optional] whether to consider boundary control via penalized Robin defaults to *False*

mergerhs [boolean, optional] whether to merge the actual rhs and the contribution from the boundary conditions into one rhs, defaults to *False*

onlymesh [boolean, optional] whether to only return *femp*, containing the mesh and FEM spaces, defaults to *False*

Returns

femp [dict]

with the keys:

- *V*: FEM space of the velocity
- *Q*: FEM space of the pressure
- *diribcs*: list of the (Dirichlet) boundary conditions
- *dbcinds*: indices of the boundary nodes
- *dbcvals*: values of the boundary nodes
- *invinds*: indices of the inner nodes
- *fv*: right hand side of the momentum equation
- *fp*: right hand side of the continuity equation
- *charlen*: characteristic length of the setup

- *nu*: the kinematic viscosity
- *Re*: the Reynolds number
- *odcoo*: dictionary with the coordinates of the domain of observation
- ***cdcoo*: dictionary with the coordinates of the domain of * *ppin*** [{int, None}]

  which dof of *p* is used to pin the pressure, typically *-1* for internal flows, and *None* for flows with outfl
  control

stokesmatsc [dict]

a dictionary of the condensed matrices:

- *M*: the mass matrix of the velocity space,
- *MP*: the mass matrix of the pressure space,
- *A*: the stiffness matrix,
- *JT*: the gradient matrix, and
- *J*: the divergence matrix
- *Jfull*: the uncondensed divergence matrix

and, if *bccontrol=True*, the boundary control matrices that weakly impose *Arob*v = Brob*u*, where

- *Arob*: contribution to *A*
- *Brob*: input operator

if mergerhs

rhsd [dict] *rhsd_vfrc* and *rhsd_stbc* merged

else

rhsd_vfrc [dict] of the dirichlet and pressure fix reduced right hand sides

rhsd_stbc [dict]

of the contributions of the boundary data to the rhs:

- *fv*: contribution to momentum equation,
- *fp*: contribution to continuity equation

Examples

femp, stokesmatsc, rhsd_vfrc, rhsd_stbc = get_sysmats(problem='drivencavity', N=10, nu=1e-2)

1.4 data_output_utils

data_output_utils.**load_or_comp**(*filestr=None*, *comprtn=None*, *comprtnargs={}*, *arraytype=None*, *debug=False*, *loadrtn=None*, *loadmsg='loaded '*, *savertn=None*, *savemsg='saved '*, *itsadict=False*, *numthings=1*)

routine for caching computation results on disc

Parameters

filestr: {string, list of strings, `None`} where to load/store the computed things, if *None* nothing is loaded or stored

arraytype: {`None`, 'sparse', 'dense'} if not None, then it sets the default routines to save/load dense or sparse arrays

itsadict: boolean, optional whether it is *python dictionary* that can be JSON serialized, overrides all other options concerning arrays

savertn: fun(), optional routine for saving the computed results, defaults to None, i.e. no saving here

debug: boolean, optional no saving or loading, defaults to *False*

data_output_utils.**output_paraview**(*V=None*, *Q=None*, *VS=None*, *fstring='nn'*, *invinds=None*, *diribcs=None*, *dbcinds=None*, *dbcvals=None*, *vp=None*, *vc=None*, *pc=None*, *sc=None*, *sname='nn'*, *ppin=None*, *t=None*, *tfilter=None*, *writeoutput=True*, *vfile=None*, *pfile=None*, *sfile=None*)

write the paraview output for a solution *(v,p)* or a scalar *s*

given as coefficients

data_output_utils.**plot_outp_sig**(*str_to_json=None*, *tmeshkey='tmesh'*, *sigkey='outsig'*, *outsig=None*, *tmesh=None*, *fignum=222*, *reference=None*, *tikzstr=None*, *compress=5*, *tikzplease=False*)

plotting output signals

data_output_utils.**save_output_json**(*datadict=None*, *fstring='unspecified_outputfile'*, *module='dolfin_navier_scipy.data_output_utils'*, *plotroutine='plot_outp_sig'*)

save output to json for postprocessing

1.5 time_int_utils

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX