Fenics Parallel Computing

Duy Duc NGUYEN

Introduction to Fenics solver

- V1 07/03/2019
- Fenics version: 2018.1.0

Introduction to Differential Algebraic Equations (DAEs)

Dof: degree of freedom (nodes on a mesh cell)
Size of a problem: small: < 10k dofs, medium: 30k < dofs, large: others

System of liner equations: $Ax = b$ System of non-liner equations: $F(x) = b$

All of them are called Differential Algebraic Equations (DAEs)

Choose solver:

LU factorization: LU is simple and robust, but only suitable for small and linear problem

Iterative method: Newton solver, Krylov solvers

- *Newton* is suitable for small or medium problem with linearity or non-linearity,
  because it computes Jabcobian matrix F' which is very expensive for large system, and for storing.
- *Krylov solvers*: GMRES, Conjuate Gradient (CG), BiCG, BiCGS, BiCGSTAB, TFQMR, IDRs, QMR, MINRES, etc
  - originally, Krylov solvers are designed for linear problem.
  - avoid to compute Jacobian matrix F', so that, they are suitable to compute large system
  - GMRES: save a basic of the Krylov subspace at each iteration. May lead to out of memory, the algorithm
    will restart using the last approximated solution as the initial guess.
  - BiCGSTAB is the 4th upgrade version of CG family, ie CG, BiCG, BiCGS and BiCGSTAB. It's dedicated to avoid storing a new basic vector of the krylov subspace at each iteration likes GMRES. However, it is not stable as GMRES
- *Newton-Krylov solvers:* Newton-GMRES, Newton-CG, Newton-BiCG etc
  - use the benefits of the Krylov solvers to solve non-linear large system
  - Recommend to use Newton-BiCGSTAB for a huge system likes 1 million dofs

C.T. Kelley – solving nonlinear equations with Newton's Method (1987)

Introduction to Differential Algebraic Equations (DAEs)

Example: Poisson problem

$$-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \text{ in } \Omega,$$
$$u(\boldsymbol{x}) = u_{\mathrm{D}}(\boldsymbol{x}), \quad \boldsymbol{x} \text{ on } \partial\Omega.$$

Strong form

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x.$$

Weak form

Numerical approximate integral
Applied boundary conditions

(Thank to Fenics)

$Ax = b$ *or* $F(x) = b$

DEAs

Fenics code for weak form

```
# For Linear problem
a = inner(grad(du), grad(v))*dx
L = f*v*dx + g*v*ds


# For Non-Linear problem
F = inner(grad(du), grad(v))*dx - f*v*dx - g*v*ds
```

High level call solver: `solve(a == L, u, bc)` `solve(F == 0, u, bc)`

Fenics will run with default setting

Built-in packages can solve DAEs in Fenics

Packages contains algorithms to solve DAEs is called: Linear algebra packages (LAP)

To check how many LAP installed in your Fenics version:

```
print(list_linear_algebra_backends())
```

```
Linear algebra backends   |  Description
--------------------------------------------------------------------------------
Eigen                     |  Template-based linear algebra  library
PETSc                     |  Powerful MPI parallel linear algebra library (default)
None
```

To know which LAP you are using:

```
print(parameters['linear_algebra_backend']) → PETSc
```

(by default)

To change your current LAP:

```
parameters['linear_algebra_backend'] = 'Eigen'
```

```
print(parameters['linear_algebra_backend']) → Eigen
```

Print all solvers:

```
print(list_linear_solver_methods())
print(list_krylov_solver_methods())
```

Built-in packages can solve DAEs in Fenics

PETSc

```
PETSc: all linear solvers
Solver method  |  Description
------------------------------------------------------------------------------
bicgstab       |  Biconjugate gradient stabilized method
cg             |  Conjugate gradient method
default        |  default linear solver
gmres          |  Generalized minimal residual method
minres         |  Minimal residual method
mumps          |  MUMPS (MUltifrontal Massively Parallel Sparse direct Solver)
petsc          |  PETSc built in LU solver
richardson     |  Richardson method
superlu        |  SuperLU
tfqmr          |  Transpose-free quasi-minimal residual method
umfpack        |  UMFPACK (Unsymmetric MultiFrontal sparse LU factorization)
None
```

```
PETSc: all krylov solvers
Krylov method  |  Description
------------------------------------------------------------------
bicgstab       |  Biconjugate gradient stabilized method
cg             |  Conjugate gradient method
default        |  default Krylov method
gmres          |  Generalized minimal residual method
minres         |  Minimal residual method
richardson     |  Richardson method
tfqmr          |  Transpose-free quasi-minimal residual method
None
```

Eigen

```
Eigen: all linear solvers
Solver method  |  Description
------------------------------------------------------------------------------
bicgstab       |  Biconjugate gradient stabilized method
cg             |  Conjugate gradient method
cholesky       |  Simplicial LDLT
cholmod        |  'CHOLMOD' sparse Cholesky factorisation
default        |  default linear solver
gmres          |  Generalised minimal residual (GMRES)
minres         |  Minimal residual
sparselu       |  Supernodal LU factorization for general matrices
umfpack        |  UMFPACK (Unsymmetric MultiFrontal sparse LU factorization)
None
```

```
Eigen: all krylov solvers
Krylov method  |  Description
------------------------------------------------------------
bicgstab       |  Biconjugate gradient stabilized method
cg             |  Conjugate gradient method
default        |  default Eigen Krylov method
gmres          |  Generalised minimal residual (GMRES)
minres         |  Minimal residual
None
```

Preconditioners

Preconditioner is matrix C that multiplied to linear DAEs to speed up the convergence rate $C^{-1}Ax = C^{-1}b$

```
print(list_krylov_solver_preconditioners())
```

PETSc

```
PETSc: all krylov preconditioners
Preconditioner   |  Description
-----------------------------------------------------------
amg              |  Algebraic multigrid
default          |  default preconditioner
hypre_amg        |  Hypre algebraic multigrid (BoomerAMG)
hypre_euclid     |  Hypre parallel incomplete LU factorization
hypre_parasails  |  Hypre parallel sparse approximate inverse
icc              |  Incomplete Cholesky factorization
ilu              |  Incomplete LU factorization
jacobi           |  Jacobi iteration
none             |  No preconditioner
petsc_amg        |  PETSc algebraic multigrid
sor              |  Successive over-relaxation
None
```

Eigen

```
Eigen: all krylov solvers
Krylov method  |  Description
-----------------------------------------------------------
bicgstab       |  Biconjugate gradient stabilized method
cg             |  Conjugate gradient method
default        |  default Eigen Krylov method
gmres          |  Generalised minimal residual (GMRES)
minres         |  Minimal residual
None
```

Introduction to solver ‘parameters’

All parameters regarding the solver, likes absolute tolerance, relatve tolerance and maximum of iteration, are set in a built-in **dictionary** in Fenics called `parameters.`

To print on terminal the parameters

```
info(parameters, True)
```

To save parameters into file

```
File('parameters.xml') << parameters
```

To read parameters from file

```
File('parameters.xml') >> parameters
```

Introduction to solver ‘parameters’

To print on terminal the parameters `info(parameters, True)`

Set of parameters: ‘dolfin’ (father of all), ‘form_compiler’, ‘krylov_solver’, ‘lu_solver’

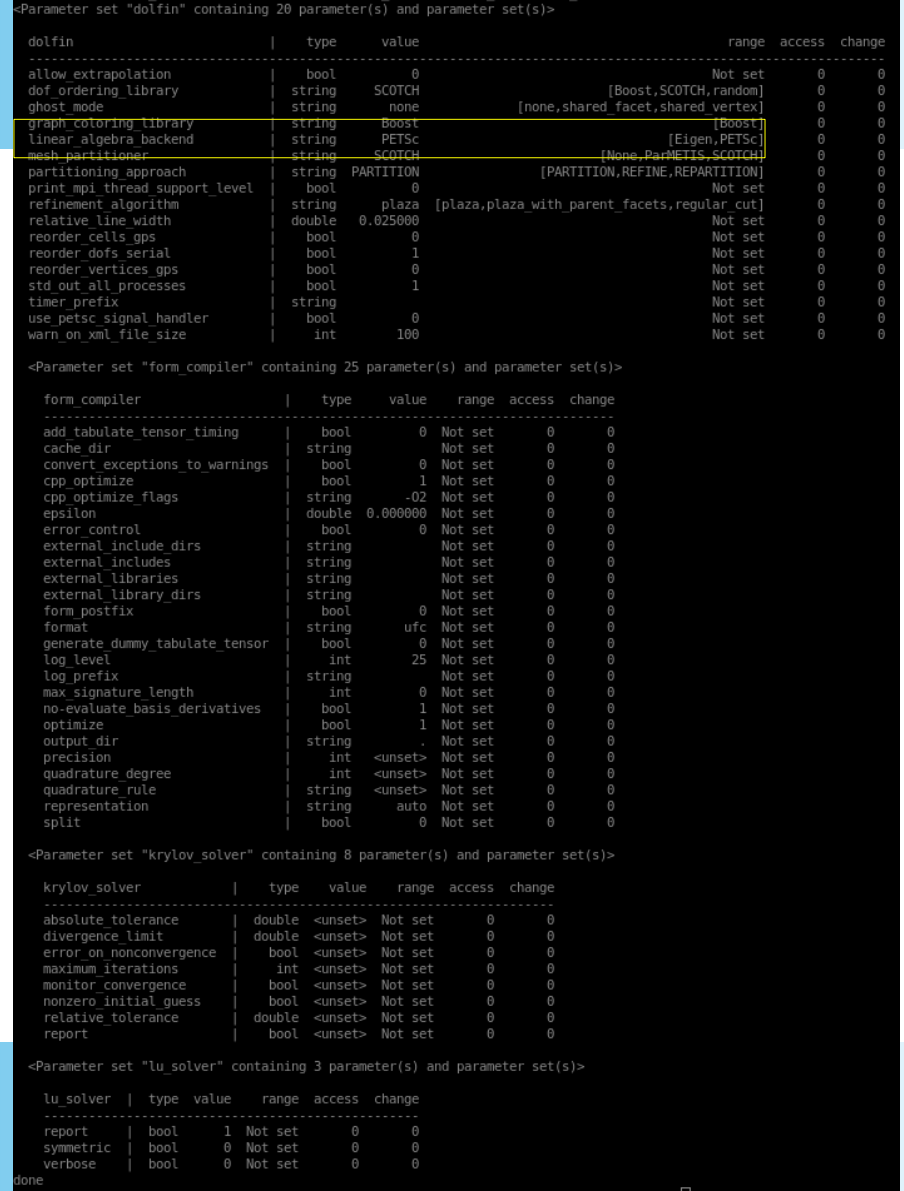
Example:
‘linear_algera_backend’ is a parameter of set ‘dolfin’

To call a set of parameters:

```
info(parameters['form_compiler'], True)
info(parameters['lu_solver'], True)
info(parameters['krylov_solver'], True)
```

To change the default setting of a parameter:

```
parameters[set_name][name] = 123 or ‘abc’
```



```
<Parameter set "dolfin" containing 20 parameter(s) and parameter set(s)>

  dolfin                          |    type      value                                                range  access  change
  ------------------------------------------------------------------------------------------------------------------------
  allow_extrapolation             |    bool          0                                              Not set       0       0
  dof_ordering_library            |  string     SCOTCH                                [Boost,SCOTCH,random]       0       0
  ghost_mode                      |  string       none                 [none,shared_facet,shared_vertex]       0       0
  graph_coloring_library          |  string      Boost                                            [Boost]       0       0
  linear_algebra_backend          |  string      PETSc                                      [Eigen,PETSc]       0       0
  mesh_partitioner                |  string     SCOTCH                          [None,ParMETIS,SCOTCH]       0       0
  partitioning_approach           |  string  PARTITION                 [PARTITION,REFINE,REPARTITION]       0       0
  print_mpi_thread_support_level  |    bool          0                                              Not set       0       0
  refinement_algorithm            |  string      plaza  [plaza,plaza_with_parent_facets,regular_cut]       0       0
  relative_line_width             |  double   0.025000                                              Not set       0       0
  reorder_cells_gps               |    bool          0                                              Not set       0       0
  reorder_dofs_serial             |    bool          1                                              Not set       0       0
  reorder_vertices_gps            |    bool          0                                              Not set       0       0
  std_out_all_processes           |    bool          1                                              Not set       0       0
  timer_prefix                    |  string                                                         Not set       0       0
  use_petsc_signal_handler        |    bool          0                                              Not set       0       0
  warn_on_xml_file_size           |     int        100                                              Not set       0       0

  <Parameter set "form_compiler" containing 25 parameter(s) and parameter set(s)>

    form_compiler                   |    type     value    range  access  change
    ------------------------------------------------------------------------------
    add_tabulate_tensor_timing      |    bool         0  Not set       0       0
    cache_dir                       |  string            Not set       0       0
    convert_exceptions_to_warnings  |    bool         0  Not set       0       0
    cpp_optimize                    |    bool         1  Not set       0       0
    cpp_optimize_flags              |  string       -O2  Not set       0       0
    epsilon                         |  double  0.000000  Not set       0       0
    error_control                   |    bool         0  Not set       0       0
    external_include_dirs           |  string            Not set       0       0
    external_includes               |  string            Not set       0       0
    external_libraries              |  string            Not set       0       0
    external_library_dirs           |  string            Not set       0       0
    form_postfix                    |    bool         0  Not set       0       0
    format                          |  string       ufc  Not set       0       0
    generate_dummy_tabulate_tensor  |    bool         0  Not set       0       0
    log_level                       |     int        25  Not set       0       0
    log_prefix                      |  string            Not set       0       0
    max_signature_length            |     int         0  Not set       0       0
    no-evaluate_basis_derivatives   |    bool         1  Not set       0       0
    optimize                        |    bool         1  Not set       0       0
    output_dir                      |  string         .  Not set       0       0
    precision                       |     int   <unset>  Not set       0       0
    quadrature_degree               |     int   <unset>  Not set       0       0
    quadrature_rule                 |  string   <unset>  Not set       0       0
    representation                  |  string      auto  Not set       0       0
    split                           |    bool         0  Not set       0       0

  <Parameter set "krylov_solver" containing 8 parameter(s) and parameter set(s)>

    krylov_solver            |    type    value    range  access  change
    -------------------------------------------------------------------
    absolute_tolerance       |  double  <unset>  Not set       0       0
    divergence_limit         |  double  <unset>  Not set       0       0
    error_on_nonconvergence  |    bool  <unset>  Not set       0       0
    maximum_iterations       |     int  <unset>  Not set       0       0
    monitor_convergence      |    bool  <unset>  Not set       0       0
    nonzero_initial_guess    |    bool  <unset>  Not set       0       0
    relative_tolerance       |  double  <unset>  Not set       0       0
    report                   |    bool  <unset>  Not set       0       0

  <Parameter set "lu_solver" containing 3 parameter(s) and parameter set(s)>

    lu_solver  |  type  value    range  access  change
    ---------------------------------------------------
    report     |  bool      1  Not set       0       0
    symmetric  |  bool      0  Not set       0       0
    verbose    |  bool      0  Not set       0       0
done
```

Introduction to solver ‘parameters’

Example: change some parameters in Krylov solver

```
prm = parameters['krylov_solver'] # for short form
prm['absolute_tolerance'] = 1E-10
prm['relative_tolerance'] = 1E-6
prm['maximum_iterations'] = 1000
```

If you change these parameters at the beginning of your code, they will **globally** applied hereafter

```
<Parameter set "krylov_solver" containing 8 parameter(s) and parameter set(s)>

  krylov_solver           |    type    value    range  access  change
  ------------------------------------------------------------------
  absolute_tolerance      |  double  <unset>  Not set       0       0
  divergence_limit        |  double  <unset>  Not set       0       0
  error_on_nonconvergence |    bool  <unset>  Not set       0       0
  maximum_iterations      |     int  <unset>  Not set       0       0
  monitor_convergence     |    bool  <unset>  Not set       0       0
  nonzero_initial_guess   |    bool  <unset>  Not set       0       0
  relative_tolerance      |  double  <unset>  Not set       0       0
  report                  |    bool  <unset>  Not set       0       0
```

Before

```
<Parameter set "krylov_solver" containing 8 parameter(s) and parameter set(s)>

  krylov_solver           |    type    value    range  access  change
  ------------------------------------------------------------------
  absolute_tolerance      |  double 0.000000  Not set       0       0
  divergence_limit        |  double  <unset>  Not set       0       0
  error_on_nonconvergence |    bool  <unset>  Not set       0       0
  maximum_iterations      |     int     1000  Not set       0       0
  monitor_convergence     |    bool  <unset>  Not set       0       0
  nonzero_initial_guess   |    bool  <unset>  Not set       0       0
  relative_tolerance      |  double 0.000001  Not set       0       0
  report                  |    bool  <unset>  Not set       0       0
```

After

Introduction to Newton algorithm

ALGORITHM 5.3.1. newton(x, F, τ)

1. $r_0 = \|F(x)\|$
2. Do while $\|F(x)\| > \tau_r r_0 + \tau_a$
   - (a) Compute $F'(x)$
   - (b) Factor $F'(x) = LU$.
   - (c) Solve $LUs = -F(x)$ $\}$ $= s_n$
   - (d) $x = x + s$
   - (e) Evaluate $F(x)$.

$$F(x)=0,$$

$$F'(x_n)\approx\frac{F(x_{n+1})-F(x_n)}{x_{n+1}-x_n},\quad x_{n+1}=x_n+F'(x_n)^{-1}\left(F(x_{n+1})-F(x_n)\right),$$

$$\|F(x_n)\|\gg\|F(x_{n+1})\|\to 0$$ So that $$x_{n+1}=x_n-F'(x_n)^{-1}F(x_n)$$

or $$x_{n+1}-x_n=-F'(x_n)^{-1}F(x_n)$$

Newton method:

$$s_n:=x_{n+1}-x_n$$
$$F'(x_n)s_n=-F(x_n)$$
$$x_{n+1}=x_n+s_n$$

Termination conditions: (Stopping criterion)

$$\|F(x)\|\leq\tau_r\|F(x_0)\|+\tau_\alpha \quad \text{and} \quad k\leqslant k_{max}$$

τ_r Relative error tolerance

τ_α Absolute error tolerance

C.T. Kelley – solving nonlinear equations with Newton's Method (1987)

Newton solver

```python
def newton_solver(V, bc, F):
        """ Description: Defind Newton algorithm
                not tested in parallel
        """
        print(">>> Call nonlinear solver: Newton <<<")
        du = TrialFunction(V)
        u = Function(V)
        F  = action(F, u)

        # Jacobian matrix
        Jab = derivative(F, u, du)

        # set solver
        problem = NonlinearVariationalProblem(F, u, bc, Jab)
        solver  = NonlinearVariationalSolver(problem)
        prm = solver.parameters

        prm['newton_solver']['absolute_tolerance'] = 1E-8
        prm['newton_solver']['relative_tolerance'] = 1E-7
        prm['newton_solver']['maximum_iterations'] = 25
        prm['newton_solver']['relaxation_parameter'] = 1.0

        #info(solver.parameters['newton_solver'], True)

        solver.solve()
```

```
newton_solver            |    type     value    range  access  change
-------------------------------------------------------------------------
absolute_tolerance       |  double  0.000000  Not set       0       0
convergence_criterion    |  string  residual  Not set       0       0
error_on_nonconvergence  |    bool         1  Not set       0       0
linear_solver            |  string   default  Not set       0       0
maximum_iterations       |     int        25  Not set       0       0
preconditioner           |  string   default  Not set       0       0
relative_tolerance       |  double  0.000000  Not set       0       0
relaxation_parameter     |  double  1.000000  Not set       0       0
report                   |    bool         1  Not set       0       0
```

No linear solver is called

Locally set parameters

Given absolute tolerance

Given relative tolerance

```
>>> Call nonlinear solver: Newton <<<
Solving nonlinear variational problem.
  Newton iteration 0: r (abs) = 3.115e+01 (tol = 1.000e-08) r (rel) = 1.000e+00 (tol = 1.000e-07)
  Newton iteration 1: r (abs) = 1.369e-14 (tol = 1.000e-08) r (rel) = 4.395e-16 (tol = 1.000e-07)
  Newton solver finished in 1 iterations and 1 linear solver iterations.
done
```

Introduction to GMRES algorithm

ALGORITHM 3.4.2. **gmresa**$(x, b, A, \epsilon, kmax, \rho)$

1. $r = b - Ax$, $v_1 = r/\|r\|_2$, $\rho = \|r\|_2$, $\beta = \rho$, $k = 0$

2. While $\rho > \epsilon\|b\|_2$ and $k < kmax$ do

   (a) $k = k + 1$

   (b) for $j = 1, \ldots, k$
   $h_{jk} = (Av_k)^T v_j$

   (c) $v_{k+1} = Av_k - \sum_{j=1}^{k} h_{jk} v_j$

   (d) $h_{k+1,k} = \|v_{k+1}\|_2$

   (e) $v_{k+1} = v_{k+1}/\|v_{k+1}\|_2$

   arnoldi

   (f) $e_1 = (1, 0, \ldots, 0)^T \in R^{k+1}$
   Minimize $\|\beta e_1 - H_k y^k\|_{R^{k+1}}$ over R^k to obtain y^k.

   (g) $\rho = \|\beta e_1 - H_k y^k\|_{R^{k+1}}$.

3. $x_k = x_0 + V_k y^k$.

Solve linear problem

$$Ax = b$$

Iteratively

GMRES solver

```python
def gmres_solver(V, bc, a, L):
        """ Description: linear solver GMRES
                can not run in parallel
        """
        print(">>> Call linear solver: GMRES <<<")
        u = Function(V)

        # define problem and solver
        problem = LinearVariationalProblem(a, L, u, bc)
        solver = LinearVariationalSolver(problem)

        # choose algorithms GMRES with ILU preconditioner
        solver.parameters['linear_solver'] = "gmres"
        solver.parameters['preconditioner'] = "ilu"
        #solver.parameters['print_rhs'] = True

        # set local parameters
        prm = solver.parameters['krylov_solver'] # short form
        prm['absolute_tolerance'] = 1E-7
        prm['relative_tolerance'] = 1E-4
        prm['maximum_iterations'] = 1000
        prm['monitor_convergence'] = True
        prm['report'] = True

        info(solver.parameters, True)

        solver.solve()

        return u
```

Print convergence message

```
>>> Call linear solver: GMRES <<<
<Parameter set "linear_variational_solver" containing 7 parameter(s) and parameter set(s)>

  linear_variational_solver  |    type  value    range  access  change
  ---------------------------------------------------------------------
  linear_solver              |  string  gmres  Not set       0       0
  preconditioner             |  string    ilu  Not set       0       0
  print_matrix               |    bool      0  Not set       0       0
  print_rhs                  |    bool      0  Not set       0       0
  symmetric                  |    bool      0  Not set       0       0

  <Parameter set "krylov_solver" containing 8 parameter(s) and parameter set(s)>

    krylov_solver            |    type     value    range  access  change
    -----------------------------------------------------------------------
    absolute_tolerance       |  double  0.000000  Not set       0       0
    divergence_limit         |  double   <unset>  Not set       0       0
    error_on_nonconvergence  |    bool   <unset>  Not set       0       0
    maximum_iterations       |     int      1000  Not set       0       0
    monitor_convergence      |    bool   <unset>  Not set       0       0
    nonzero_initial_guess    |    bool   <unset>  Not set       0       0
    relative_tolerance       |  double  0.000100  Not set       0       0
    report                   |    bool         1  Not set       0       0

  <Parameter set "lu_solver" containing 3 parameter(s) and parameter set(s)>

    lu_solver  |  type  value    range  access  change
    ------------------------------------------------
    report     |  bool      1  Not set       0       0
    symmetric  |  bool      0  Not set       0       0
    verbose    |  bool      0  Not set       0       0
```

Preconditioner 'ilu' can not run in parallel, use 'hypre_euclid' instead

```
Solving linear variational problem.
  Solving linear system of size 549 x 549 (PETSc Krylov solver).
done
```

```
Solving linear variational problem.
  Solving linear system of size 549 x 549 (PETSc Krylov solver).
  0 KSP preconditioned resid norm 1.087833032130e+01 true resid norm 8.944633496442e+00 ||r(i)||/||b|| 1.000000000000e+00
  1 KSP preconditioned resid norm 3.177442391705e+00 true resid norm 3.237513682686e+00 ||r(i)||/||b|| 3.619504012069e-01
  2 KSP preconditioned resid norm 1.646247475198e+00 true resid norm 1.709854243620e+00 ||r(i)||/||b|| 1.911597880786e-01
  3 KSP preconditioned resid norm 1.133144175467e+00 true resid norm 1.042557609266e+00 ||r(i)||/||b|| 1.165567722457e-01
```

Introduction to Newton-GMRES algorithm

ALGORITHM 6.2.1. `fdgmres`$(s, x, F, h, \eta, kmax, \rho)$

1. $s = 0$, $r = -F(x)$, $v_1 = r/\|r\|_2$, $\rho = \|r\|_2$, $\beta = \rho$, $k = 0$
2. While $\rho > \eta\|F(x)\|_2$ and $k < kmax$ do
   (a) $k = k + 1$
   (b) $v_{k+1} = D_h F(x : v_k)$
   for $j = 1, \ldots k$
   i. $h_{jk} = v_{k+1}^T v_j$
   ii. $v_{k+1} = v_{k+1} - h_{jk} v_j$
   (c) $h_{k+1,k} = \|v_{k+1}\|_2$
   (d) $v_{k+1} = v_{k+1}/\|v_{k+1}\|_2$
   (e) $e_1 = (1, 0, \ldots, 0)^T \in R^{k+1}$
   Minimize $\|\beta e_1 - H_k y^k\|_{R^{k+1}}$ to obtain $y^k \in R^k$.
   (f) $\rho = \|\beta e_1 - H_k y^k\|_{R^{k+1}}$.
3. $s = V_k y^k$. output

Inexact termination condition

ALGORITHM 6.3.1. `nsolgm`(x, F, τ, η)

1. $r_c = r_0 = \|F(x)\|_2/\sqrt{N}$
2. Do while $\|F(x)\|_2/\sqrt{N} > \tau_r r_0 + \tau_a$
   (a) Select η.
   (b) `fdgmres`(s, x, F, η) $= s_n$
   (c) $x = x + s$
   (d) Evaluate $F(x)$
   (e) $r_+ = \|F(x)\|_2/\sqrt{N}, \sigma = r_+/r_c, r_c = r_+$
   (f) If $\|F(x)\|_2 \le \tau_r r_0 + \tau_a$ exit.

Termination condition

GMRES is originally designed for solving linear problem

$$Ax = b$$

Newton algorithm uses GMRES to find s_n

$$F'(x)s = F(x)$$

Forward difference GMRES algorithm

C.T. Kelley – solving nonlinear equations with Newton's Method (1987)

Newton-GMRES solver

```
def newton_gmres_solver(V, bc, F):
        print(">>> Call nonlinear solver: Newton-GMRES <<<")
        du = TrialFunction(V)
        u = Function(V)
        F  = action(F, u)

        # Jacobian matrix
        Jab = derivative(F, u, du)

        # set problem and solver
        problem = NonlinearVariationalProblem(F, u, bc, Jab)
        solver  = NonlinearVariationalSolver(problem)
        prm = solver.parameters

        # set Newton solver
        prm['newton_solver']['absolute_tolerance'] = 1E-8
        prm['newton_solver']['relative_tolerance'] = 1E-7
        prm['newton_solver']['maximum_iterations'] = 50
        prm['newton_solver']['relaxation_parameter'] = 1.0
        prm['newton_solver']['linear_solver'] = 'gmres'
        prm['newton_solver']['preconditioner'] = 'ilu'

        #info(prm['newton_solver'], True)

        # set GMRES solver
        prm['newton_solver']['krylov_solver']['absolute_tolerance'] = 1E-9
        prm['newton_solver']['krylov_solver']['relative_tolerance'] = 1E-7
        prm['newton_solver']['krylov_solver']['maximum_iterations'] = 1000

        #info(prm['newton_solver'], True)
        solver.solve()
```

Newton-GMRES solver

```
>>> Call nonlinear solver: Newton-GMRES <<<
<Parameter set "newton_solver" containing 11 parameter(s) and parame

  newton_solver            |    type     value    range  access  cha
  -----------------------------------------------------------------
  absolute_tolerance       |  double  0.000000  Not set       0       0
  convergence_criterion    |  string  residual  Not set       0       0
  error_on_nonconvergence  |    bool         1  Not set       0       0
  linear_solver            |  string     gmres  Not set       0       0
  maximum_iterations       |     int        50  Not set       0       0
  preconditioner           |  string       ilu  Not set       0       0
  relative_tolerance       |  double  0.000000  Not set       0       0
  relaxation_parameter     |  double  1.000000  Not set       0       0
  report                   |    bool         1  Not set       0       0

  <Parameter set "krylov_solver" containing 8 parameter(s) and parameter set(s)>

    krylov_solver            |    type     value    range  access  change
    -------------------------------------------------------------------
    absolute_tolerance       |  double  0.000000  Not set       0       0
    divergence_limit         |  double   <unset>  Not set       0       0
    error_on_nonconvergence  |    bool   <unset>  Not set       0       0
    maximum_iterations       |     int      1000  Not set       0       0
    monitor_convergence      |    bool   <unset>  Not set       0       0
    nonzero_initial_guess    |    bool   <unset>  Not set       0       0
    relative_tolerance       |  double  0.000000  Not set       0       0
    report                   |    bool   <unset>  Not set       0       0

  <Parameter set "lu_solver" containing 3 parameter(s) and parameter set(s)>

    lu_solver  |  type  value    range  access  change
    ---------------------------------------------
    report     |  bool      1  Not set       0       0
    symmetric  |  bool      0  Not set       0       0
    verbose    |  bool      0  Not set       0       0
```

```
>>> Call nonlinear solver: Newton-GMRES <<<
Solving nonlinear variational problem.
  Newton iteration 0: r (abs) = 3.115e+01 (tol = 1.000e-08) r (rel) = 1.000e+00 (tol = 1.000e-07)
  Newton iteration 1: r (abs) = 1.983e-06 (tol = 1.000e-08) r (rel) = 6.368e-08 (tol = 1.000e-07)
  Newton solver finished in 1 iterations and 5 linear solver iterations.
```

Newton algorithm takes 2 iterations

GMRES takes 6 iterations

Relative tolerance is smaller than the given relative tolerant. Stopping condition is satisfied.

Good lecturers and references

http://users.ices.utexas.edu/~noemi/teaching.html

https://fenicsproject.org/pub/course/lectures/2017-nordic-phdcourse/

http://www.math.pitt.edu/~sussmanm/

FEniCS tutorial (Python):http://fenicsproject.org/documentation/tutorial/
Examples:http://fenicsproject.org/_static/tutorial/fenics_tutorial_examples.tar.gz
FEniCS presentations:http://fenicsproject.org/pub/presentations/
FEniCS course:http://fenicsproject.org/pub/course/lectures/
Documented DOLFIN demos (Python):note: the demos are already installed on your system, see fo ex:/Applications/FeniCS.app/Contents/Resources/share/dolfin/demohttp://fenicsproject.org/documentation/dolfin/1.1.0/python/demo/index.html
Basic classes and functions in DOLFIN (Python):
http://fenicsproject.org/documentation/dolfin/1.1.0/python/quick_reference.html
All classes and functions in DOLFIN (Python):
http://fenicsproject.org/documentation/dolfin/1.1.0/python/genindex.html

Questions on forum

Use subprocess: object oriented programming
https://fenicsproject.org/qa/10701/on-using-mpi/

Marked mesh with GMSH tag does not work in parallel:
https://fenicsproject.org/qa/5337/importing-marked-mesh-for-parallel-use/?show=5344#a5344

https://fenicsproject.org/qa/5864/get-access-to-entire-vector-in-parallel/

https://fenicsproject.org/qa/4227/going-parallel-distributed/

https://fenicsproject.org/qa/13142/assemble-system-parallel-and-store-complete-sparse-matrix/

https://fenicsproject.org/qa/3025/solving-a-problem-in-parallel-with-mpi-&-petsc/

Parallel tutorial

https://fenicsproject.org/qa/3154/improve-performance-for-solving-coupled-system-in-parallel/
https://fenicsproject.org/qa/3324/using-petsc-direct-solver-in-parallel/
https://fenicsproject.org/qa/12217/parallel-solver-calls/

https://fenicsproject.org/qa/8459/what-is-the-simplest-way-to-use-mpi/

http://www.math.pitt.edu/~sussmanm/3040Summer14/fenicsI.pdf

https://media.readthedocs.org/pdf/fenics-handson/latest/fenics-handson.pdf

https://fenicsproject.org/qa/10068/solving-linear-system-in-parallel/

http://andy.terrel.us/papers_and_talks/FEniCS08Tutorial.pdf

https://www.mcs.anl.gov/petsc/documentation/tutorials/ACTS2006/ACTS2006.pdf

https://cse.buffalo.edu/~knepley/presentations/TutorialFEniCS2008.pdf