Symbolic Integrators¶

Symbolic integrators can be used to define arbitrary (multi-)physical problems. The variational formulation of the (non-)linear problem can be implemented in a very natural way. Examples of the usage of these integrators can i.e. be found in Navier Stokes Equation, Magnetic Field or Nonlinear elasticity.

The finite element space provides placeholder coefficient functions by the methods `TestFunction` and `TrialFunction` . They have implemented canonical derivatives and traces of the finite element space and insert the basis functions of the FESpace during assembling of the system matrices.

Linear Problems

For linear problems we use the function `Assemble` of the `BilinearForm` to assemble the matrix and vector. For example for the H_{curl} linear problem

$$\int_\Omega \mu^{-1} \nabla \times u \cdot \nabla \times v + 10^{-6} u \cdot v dx = \int_C \begin{pmatrix} y \\ -x \\ 0 \end{pmatrix} . v dx \tag{1}$$

from example Magnetic Fields we have to define the space

In [ ]:
```
a = BilinearForm(fes, symmetric=True)
a += nu*curl(u)*curl(v)*dx + 1e-6*nu*u*v*dx

a.Assemble()
```

as well as the LinearForm

The argument of the symbolic integrator must be a coefficient function depending linearily on the test and trial function.

BilinearForm.Assemble(self: ngsolve.comp.BilinearForm, reallocate: bool = Fal

Assemble the bilinear form.

Parameters:

reallocate : bool

input reallocate

Nonlinear Problems

If your left hand side of the variational formulation is nonlinear there are multiple ways to get a discretisation, depending on what you want.

Apply

Here is some code! The function Apply applies the formulation to the given BaseVector . You can get a BaseVector form your GridFunction with GridFunction.vec . The output vector can be created with BaseVector.CreateVector .

BilinearForm.Apply(self: ngsolve.comp.BilinearForm, x: ngsolve.la.BaseVector, y

Applies a (non-)linear variational formulation to x and stores the result in y.

Parameters:

x : ngsolve.BaseVector

input vector

y : ngsolve.BaseVector

output vector

AssembleLinearization

For a variational formulation

$$\int_\Omega f(u)vdx \tag{2}$$

the method AssembleLinearization computes

$$\int_\Omega f^{'}(u_{lin})uvdx \tag{3}$$

with automatic differentiation of $f(u)$ and an input BaseVector u_{in}.

BilinearForm.AssembleLinearization(self: ngsolve.comp.BilinearForm, ulin: ng

Computes linearization of the bilinear form at given vecor.

Parameters:

ulin : ngsolve.la.BaseVector

Assemble

You can do your own linearization as well using Assemble and a GridFunction as a CoefficientFunction in your integrator. Let gfu_old be this gridfunction then

In [ ]:
```
a = BilinearForm(fes)
a += SymbolicBFI(gfu_old * u * v)
```

will be a linearization for

$$\int_\Omega u^2vdx \tag{4}$$

Every time you call Assemble the bilinearform is updated with the new values of the GridFunction.

Symbolic Energy

`SymbolicEnergy` can be used to solve a minimization problem. In this tutorial we show how to solve the nonlinear problem

$$\min_{u \in V} 0.05 \nabla u + u^4 - 100u \tag{5}$$

In [ ]:
```
a = BilinearForm (V, symmetric=False)
a += Variation( (0.05*grad(u)*grad(u) + u*u*u*u - 100*u)*dx )
```

from the `GridFunction` we can create new `BaseVector` :

With this we can use `AssembleLinearization` to do a Newton iteration to solve the problem:

In [ ]:
```
for it in range(20):
    print ("Newton iteration", it)
    print ("energy = ", a.Energy(u.vec))

    a.Apply (u.vec, res)
    a.AssembleLinearization (u.vec)
    inv = a.mat.Inverse(V.FreeDofs())
    w.data = inv * res
    print ("w*r =", InnerProduct(w,res))

    u.vec.data -= w
    Redraw()
```