Vectors and matrices¶

November 1, 2022

1 Vectors and matrices¶

NGSolve contains two different implementations of linear algebra: One deals with dense matrices which are typically small, the other one with typically large sparse matrices and linear operators as needed in the finite element method.

1.1 Large Linear Algebra

Grid-functions, bilinear-forms and linear-forms create vectors and matrices. You can query them using the u.vec or bfa.mat attributes. You can print them via print (u.vec), or set values gf.vec[0:10] = 3.7.

You can create new vectors of the same size and the same element type via

[ ]:
```
vu = u.vec
help = vu.CreateVector()
```

You can perform vector-space operations

[ ]:
```
help.data = 3 * vu
help.data += mat * vu
print ("(u,h) = ", InnerProduct(help, vu))
```

There are a few things to take care of:

- Use the .data attribute to write to an existing vector. The expression help = 3 * vu will redefine the object help to something like the symbolic expression product of scalar times vector.
- You can combine certain operations (e.g. help.data = 3 * u1 + 4 * u2 + mat * u4), but not arbitrary operations (as help.data = mat * (u1+u2) or help.data = mat1 * mat2 * u). The ratio behind is that the operations must be computable without allocating temporary vectors.

You can also work with NGSolve-Data from numpy/scipy, see also here `ngspy-numpy` ## Small Linear Algebra

With x = Vector(5) and m = Matrix(3,5) you create a vector and a matrix. You can access elements with brackets, and perform linear algebra operations. y = m * x defines a new vector y.

ngsolve provides elementary matrix-vector operations. For other operations, we recommend to use the numpy package. With m.NumPy() you get a multi-dimensional numpy array sharing the memory of the matrix m.