NewtonImplicitSolver

April 24, 2023

1   NewtonImplicitSolver

```
[1]: from casadi import *
     from numpy import *
     from pylab import *
```

We will investigate the working of rootfinder with the help of the parametrically exited Duffing equation. ! $\ddot{u} + \dot{u} - \epsilon(2\mu\dot{u} + \alpha u^3 + 2ku\cos(\Omega t))$ with $\Omega = 2 + \epsilon\sigma$. \

The first order solution is $u(t) = a\cos(\frac{1}{2}\Omega t - \frac{1}{2}\gamma)$ with the modulation equations: \ $\frac{da}{d\epsilon t} = -\left[\mu a + \frac{1}{2}ka\sin\gamma\right]$ \ $a\frac{d\gamma}{d\epsilon t} = -\left[-\sigma a + \frac{3}{4}\alpha a^3 + ka\cos\gamma\right]$ \

We seek the stationary solution to these modulation equations.

Parameters

```
[2]: eps    = SX.sym("eps")
     mu     = SX.sym("mu")
```

```
alpha = SX.sym("alpha")
k     = SX.sym("k")
sigma = SX.sym("sigma")
params = [eps,mu,alpha,k,sigma]
```

Variables

```
[3]: a     = SX.sym("a")
     gamma = SX.sym("gamma")
```

Equations

```
[4]: res0 = mu*a+1.0/2*k*a*sin(gamma)
     res1 = -sigma * a + 3.0/4*alpha*a**3+k*a*cos(gamma)
```

Numerical values

```
[5]: sigma_ = 0.1
     alpha_ = 0.1
     k_     = 0.2
     params_ = [0.1,0.1,alpha_,k_,sigma_]
```

We create a NewtonImplicitSolver instance

```
[6]: f=Function("f", [vertcat(a,gamma),vertcat(*params)],[vertcat(res0,res1)])
     opts = {}
     opts["abstol"] = 1e-14
     opts["linear_solver"] = "csparse"
     s=rootfinder("s", "newton", f, opts)
```

Initialize $[a,\gamma]$ with a guess and solve

```
[7]: x_ = s([1,-1], params_)
```

```
[8]: print("Solution = ", x_)
```

```
Solution =  [1.1547, -1.5708]
```

Compare with the analytic solution:

```
[9]: x = [sqrt(4.0/3*sigma_/alpha_),-0.5*pi]
     print("Reference solution = ", x)
```

```
Reference solution =  [1.1547005383792515, -1.5707963267948966]
```

We show that the residual is indeed (close to) zero

```
[10]: residual = f(x_, params_)
      print("residual = ", residual)
```

```
residual =  [1.29063e-15, 2.62584e-15]
```

```
[11]:  for i in range(1):
           assert(abs(x_[i]-x[i])<1e-6)
```

Solver statistics

```
[12]:  print(s.stats())
```

```
{'iter_count': 29, 'n_call_g': 0, 'n_call_jac_f_z': 0, 'return_status':
'success', 'success': True, 't_proc_g': 0.0, 't_proc_jac_f_z': 0.0, 't_wall_g':
0.0, 't_wall_jac_f_z': 0.0, 'unified_return_status': 'SOLVER_RET_UNKNOWN'}
```