morfem

Szymon Knopp

Dec 14, 2022

CONTENTS

morfem(*domain*, *a0*, *a1*, *a2*, *b*, *t_a0=lambda t: 1*, *t_a1=lambda t: t*, *t_a2=lambda t: t**2,t_b=lambda t: t*)

Solves finite element method problem defined as:
(`t_a0` * `a0` + `t_a1` * `a1` + `t_a2` * `a2`) *x* = `t_b` * `b`
using model order reduction algorithms.

Parameters

domain
[*vector, shape (I)*] Ordered set of *I* domain points `t`, that the problem should be solved for.

a0
[*sparse csc array, shape (N, N)*] First part of a system matrix.

a1
[*sparse csc array, shape (N, N)*] Second part of a system matrix.

a2
[*sparse csc array, shape (N, N)*] Third part of a system matrix.

b
[*sparse csc array, shape (N, M)*] Part of an impulse vector.

t_a0
[*(float) -> float*] Function returning coefficient for `a0`. `t` -> `1` by default.

t_a1
[*(float) -> float*] Function returning coefficient for `a1`. `t` -> `t` by default.

t_a2
[*(float) -> float*] Function returning coefficient for `a2`. `t` -> `t**2` by default.

t_b
[*(float) -> float*] Function returning coefficient for `b`. `t` -> `t` by default.

Returns

(x, q, a0_r, a1_r, a2_r, b_r)
[*tuple of ndarrays*]

- **x** - *shape (I, Nr, M)*, array of problem solutions, where `x[n]` is a solution for the *n*-th point in the `domain`
- **q** - *shape (N, Nr)* projection base, product of model order reduction algorithm
- **a0_r** - *shape (Nr, Nr)*, reduced `a0`, equal to `q.T @ a0 @ q`
- **a1_r** - *shape (Nr, Nr)*, reduced `a1`, equal to `q.T @ a1 @ q`
- **a2_r** - *shape (Nr, Nr)*, reduced `a2`, equal to `q.T @ a2 @ q`
- **b_r** - *shape (Nr, M)*, reduced `b`, equal to `q.T @ b`

Example

Given the finite element method problem defined as *(G - t2 C)X = tB*, it can be solved by calling:

```
x, q, g_r, _, c_r, b_r = morfem(domain, G, csc_array(G.shape), C, B, t_
↪a2=lambda t: -t**2)
```

Equivalent calls include, (among others):

```
morfem(domain, G, C, csc_array(G.shape), B, t_a1=lambda t: -t**2)
morfem(domain, G, C, csc_array(G.shape), B, t_a0=lambda t: 1, t_a1=lambda t: -t**2, t_b=lambda t: t)
```

If, for example *E* is needed for all the points in domain, where *E_t = t(X_transposed)B*, it can be calculated as:

```
e = np.zeros((domain.size, x.shape[1], b_r.shape[1]))
for i in range(domain.size):
    e[i] = domain[i] * x[i].T * b_r
```