Sparse computations and solving linear systems - a practical introduction

02623 - The Finite Element Method for Partial Differential Equations

Authors: Allan Peter Engsig-Karup

Scientific Computing Section

$$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$$

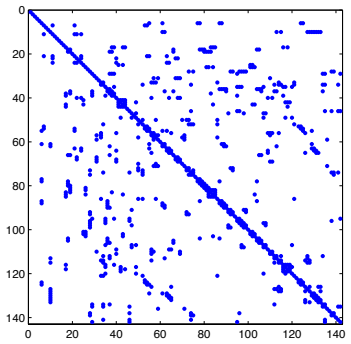
$$\int_a^b \Theta \quad \sqrt{17} \quad \delta e^{i\pi} = -1 \quad \{2.7182818284\} \quad \lambda \quad \chi^2 \quad \sum !$$

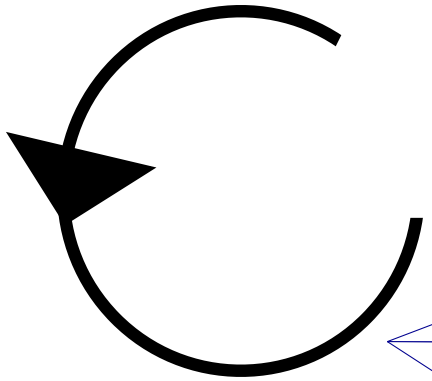
DTU Compute
Department of Applied Mathematics and Computer Science

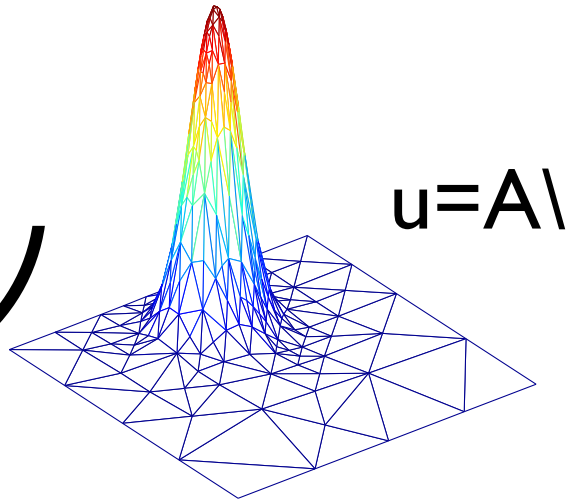
Overview

$$\nabla^2 u(x,y) = f(x,y), \quad (x,y) \in \Omega$$

Au=b







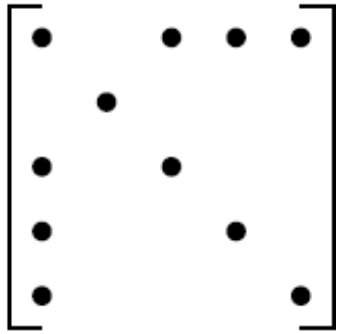
u=A\b

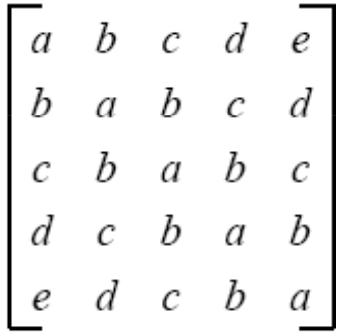
- Sparse computations in Matlab
- Solution of linear system of equations

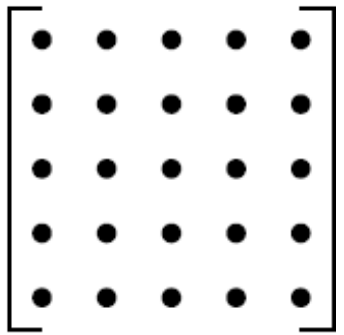
Sparse computations in Matlab

Matrix types

- A *sparse* matrix is a matrix with enough zeroes to take advantage of it [J. H. Wilkinson].
- A *structured* matrix has enough structure so that it is worthwhile to use it.
- A *dense* matrix is neither sparse nor structured.







Matrix storage in Matlab

- The only data type in Matlab is an array (=matrix)
- The same matrix can be represented in different ways
- In Matlab most details are hidden from the user and provided as "black box" tools.
- The choice of representation have different storage requirements
- In numerical algorithms the storage choice may affect the computational efficiency of computations

A basic understanding of this can increase performance and productivity...
...significantly!

Data structures for matrices

Full

- Storage: Array of real (or complex) numbers
- Memory:
  nrows*ncols

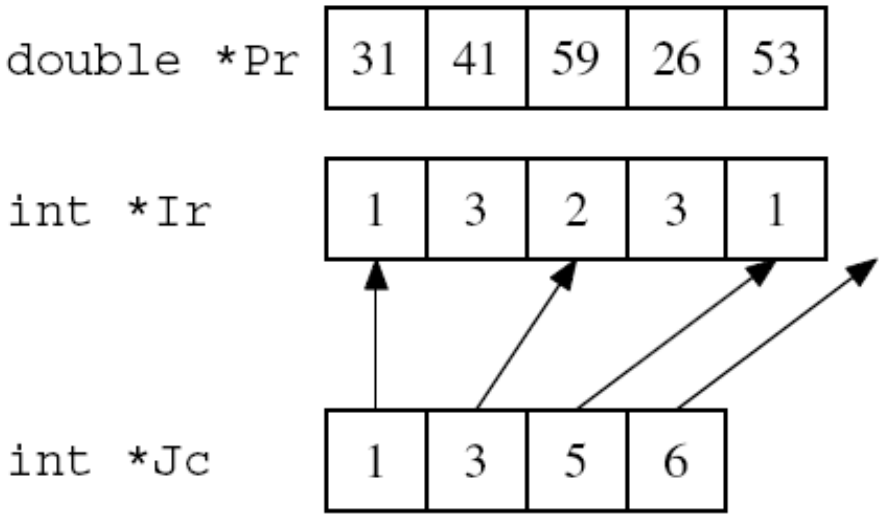
31	0	53
0	59	0
41	26	0

`double *A`

Sparse

- Compressed column storage
- Memory:
  approx. 1.5*nnz+0.5*ncols
- Note: Insertion of new nonzeros very expensive due to required memory re-allocation

double = 8 bytes, int32 = 4 bytes



Remark: The Jc vector tells what elements starts the next column. Fx. element 3 is the first element in column 2.

Dense Matrix Functions

>> Help elmat;

- Elementary matrices: **eye, zeros, ones, rand**, **diag**,...
- Basic array information: **size, length, numel,** ...
- Working with matrices: **reshape, repmat**, **spy**,...
- Reordering algorithms: **symrcm, colamd**, **randperm**,...
- Linear algebra: **eig**, **svd**, **lu**, **cond**,...
- Linear equations (iterative methods): **pcg**, **cgs**, **gmres**, ...
- Operations on graphs (trees): **treelayout**, **etree**, ...
- Specialized matrices: **compan, hankel, magic, pascal,** ...

Sparse Matrix Functions

>> Help sparfun;

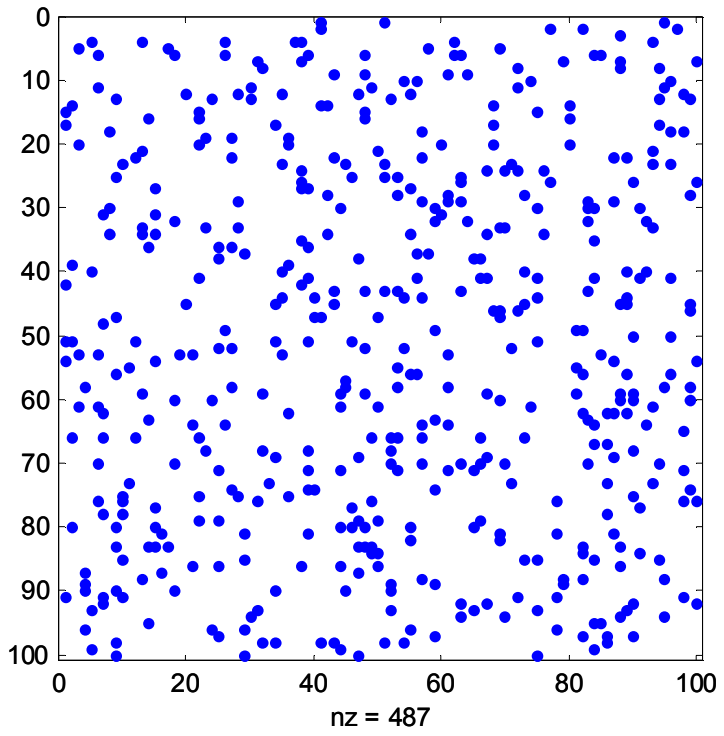
- Elementary sparse matrices: **speye**, **sprand**, **sprandn**,...
- Full to sparse conversion: **sparse**, **full**, **find**,...
- Working with sparse matrices: **nnz**, **issparse**, **spy**,...
- Reordering algorithms: **colamd**, **colperm**, **randperm**,...
- Linear algebra: **eigs**, **svds**, **luinc**, **condest**,...
- Linear equations (iterative methods): **pcg**, **cgs**, **gmres**, ...
- Operations on graphs (trees): **treelayout**, **etree**, ...
- Miscellaneous: **symbfact**, **spparms**, **spaugment**, ...

Sparse matrices

Example:

A 100x100 matrix with 10.000 elements and few nonzeros.

```
>> sp = sprand(100,100,0.05);
>> spy(sp)
```



Total number of nonzeros

```
>> nnz(sp)
ans =
   487
```

corresponding to a fill of 4.87 %.

Data structures for matrices

Conversion between full/sparse

```
>> As = sprand(100,100,0.05);
>> Af = full(As);
```

Matlab workspace memory usuage

```
>> whos As Af
  Name        Size              Bytes  Class     Attributes

  Af        100x100             80000  double
  As        100x100              6224  double    sparse
```

Extraction of sparse matrix in ”Coordinate format”

```
>> [i,j,s] = find(As);
```

Note: Not compressed column storage format in user-output!

Creation of sparse matrix directly

```
>>As = sparse(i,j,s);
```

Matlab sparse design considerations

- Most operations should give the same results for sparse and full matrices (exception **issparse**)
- Sparse matrices are never created automatically, but once created they propagate (few exceptions S*F=F, S\F=F and some more)
- Performance is important – but usability, simplicity, completeness , and robustness are more important
- Storage for a sparse matrix should scale linearly with the number of nonzeros, O(nonzeros)
- Time for a sparse operation should scale linearly with the number of arithmetic operations, O(flops)

Reference: Gilbert, Moler &Schreiber (1992) ”Sparse matrices in Matlab: Design and impementation”.

Sparse arrays

- A sparse matrix can be used for enhancing efficiency
  - Reduce storage requirements significantly
  - Carry out operations fast for arrays with few nonzero elements

Example: Dense vs. Sparse Matrix-Matrix product

```
>> A=magic(100); I = eye(100); tic, A*I; toc
Elapsed time is 0.001782 seconds.

>> A=magic(100); I = speye(100); tic, A*I; toc
Elapsed time is 0.000562 seconds.
```

Sparse matrix-matrix operation pays off in Matlab

Sparse arrays

Example: Construct a sparse unity matrix directly

```
m = 3;
i = zeros(1,m); j = zeros(1,m); s = zeros(1,m);
for n = 1 : m
    i(n) = n;
    j(n) = n;
    s(n) = 1;
end
U = sparse(i,j,s);
```

Coordinate (COO) storage!

Note: The ”Short-cut” using the Matlab utility function for this matrix is

```
>> U = speye(3);
```

General rule: Generally, it is most efficient to assemble a given sparse matrix directly from the storage vectors i,j and s.

Why not just stick with sparse matrices?

The data structure, as well as the computations, involve some overhead when a sparse matrix is really not sparse.

Example: A full matrix stored as sparse

```
>> A=magic(32);
>> Asp=sparse(A);
>> whos
  Name      Size            Bytes  Class     Attributes

  A        32x32            8192  double
  Asp      32x32            12420  double    sparse

>> n=1001;A=magic(n);Asp=sparse(A);b=rand(n,1);
>> tic, x=A*b;toc
Elapsed time is 0.002706 seconds.
>> tic, x=Asp*b;toc
Elapsed time is 0.092124 seconds.
```

Sparse operations

All operations on dense arrays also work on sparse matrices.

```
>> S = sprand(5,5,1);     F = rand(5);
>> StimesS = S*S;         StimesF = S*F;
>> SplusS = S+S;          SplusF = S+F;
>> whos
  Name        Size          Bytes  Class     Attributes

  F           5x5             200  double
  S           5x5             180  double    sparse
  SplusS      5x5             276  double    sparse
  SplusF      5x5             200  double
  StimesS     5x5             276  double    sparse
  StimesF     5x5             200  double
```

General rule: the result is sparse if both operands are sparse

Sparse computations in Python

Dense Matrix Functions

Python modules: **numpy**, **scipy**

- Elementary matrices: **numpy.eye**, **numpy.zeros**, **numpy.ones**, **numpy.random.rand**, **numpy.diag, …**
- Basis array information: **numpy.shape**, **numpy.size**, **len**, …
- Working with matrices: **numpy.reshape**, **numpy.tile**, …
- Reordering algorithms: **scipy.sparse.csgraph.reverse_cuthill_mckee**, **numpy.random.permulation**,…
- Linear algebra: **numpy.linalg.eig, numpy.linalg.svd**, **scipy.linalg.lu**, **numpy.linalg.cond**,…
- Linear equations (iterative methods): **scipy.sparse.linalg.cg**, **scipy.sparse.linalg.cgs**, **scipy.sparse.linalg.gmres**,…
- Specialized matrices: **scipy.linalg.companion**, **scipy.linalg.hankel**, **scipy.linalg.magic**, **scipy.linalg.pascal**,…

Sparse Matrix Functions

Python modules: **numpy**, **scipy**

- Elementary sparse matrices: **scipy.sparse.eye**, **scipy.sparse.rand**, **scipy.sparse.randn**, …
- Full to sparse conversion: **scipy.sparse.csr_matrix**, **scipy.sparse.csc_matrix**, **.todense()**, **.toarray()**, **scipy.sparse.find**, …
- Working with sparse matrices: **nnz**, **scipy.sparse.issparse**, **scipy.sparse.spy**, …
- Reordering algorithms: **scipy.sparse.csgraph.colamd**, **scipy.sparse.csgraph.reverse_cuthill_mackee**, …
- Linear algebra: **scipy.sparse.linalg.eigs**, **scipy.sparse.linalg.eigsh**, **scipy.sparse.linalg.svds**, **scipy.sparse.linalg.spilu**, **scipy.sparse.linalg.condest**, …
- Linear equations (iterative methods): **scipy.sparse.linalg.cg**, **scipy.sparse.linalg.cgs**, **scipy.sparse.linalg.gmres**, …
- Miscellaneous: **scipy.sparse.linalg.splu**, **scipy.sparse.linalg.spilu**, **scipy.sparse.linalg.spmv**, …

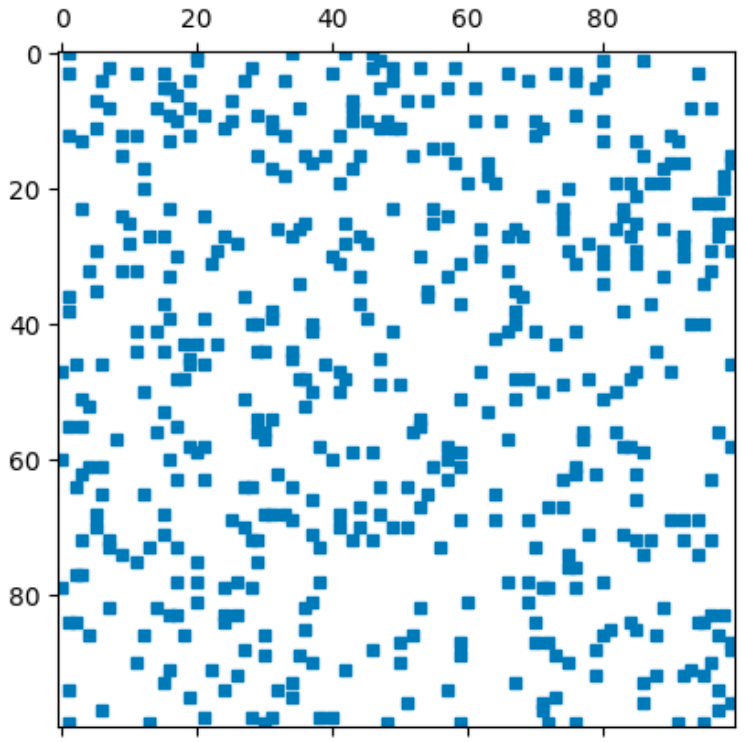
Sparse matrices

Example:

A 100x100 matrix with 10.000 elements and few nonzeros.

```
import numpy as np
import scipy.sparse as sparse
import matplotlib.pyplot as plt

# Create a 100x100 sparse matrix with approximately 5% non-zero entries
sp = sparse.rand(100, 100, density=0.05, format='coo')

# Plot the non-zero entries
plt.spy(sp, markersize=5)
plt.show()

# Output the number of non-zero entries
nnz = sp.nnz
print(f"Total number of nonzeros: {nnz}")
```

```
Total number of nonzeros: 500
corresponding to a fill of 5.00 %.
```



Data structures for matrices

Conversion between full/Sparse:

```
import numpy as np
import scipy.sparse as sparse
import sys

As = sparse.rand(100, 100, density=0.05, format='coo')  # Sparse matrix
Af = As.toarray()  # Dense matrix conversion
```

Python workspace memory usage:

```
print(f"Size of dense matrix in bytes: {sys.getsizeof(Af)}")
print(f"Size of sparse matrix in bytes: {sys.getsizeof(As)}")
```

```
Size of dense matrix in bytes: 80128
Size of sparse matrix in bytes: 48
```

Extraction of sparse matrix in “coordinate” format:

```
rows, cols = As.row, As.col
vals = As.data        # Note: In Python, the 'data' attribute gives the non-zero values
```

Creation of sparse matrix directly:

```
As_direct = sparse.coo_matrix((vals, (rows, cols)))
```

Sparse arrays

Compare full/Sparse:

```
import numpy as np
import scipy.sparse as sparse
import time

# Dense matrix multiplication
A = np.array([[i+j*100 for i in range(100)] for j in range(100)])  # Substitute for MATLAB's magic(100)
I_dense = np.eye(100)
start_time = time.time()
result_dense = A.dot(I_dense)
elapsed_dense = time.time() - start_time
print(f"Dense matrix multiplication took {elapsed_dense:.5f} seconds.")

# Sparse matrix multiplication
I_sparse = sparse.eye(100, format='csc')  # Using Compressed Sparse Column format for efficiency
start_time = time.time()
result_sparse = A.dot(I_sparse)
elapsed_sparse = time.time() - start_time
print(f"Sparse matrix multiplication took {elapsed_sparse:.5f} seconds.")
```

```
Dense matrix multiplication took 0.00505 seconds.
Sparse matrix multiplication took 0.27074 seconds...!!
```

Significantly more expensive than Matlab

Python code profiling

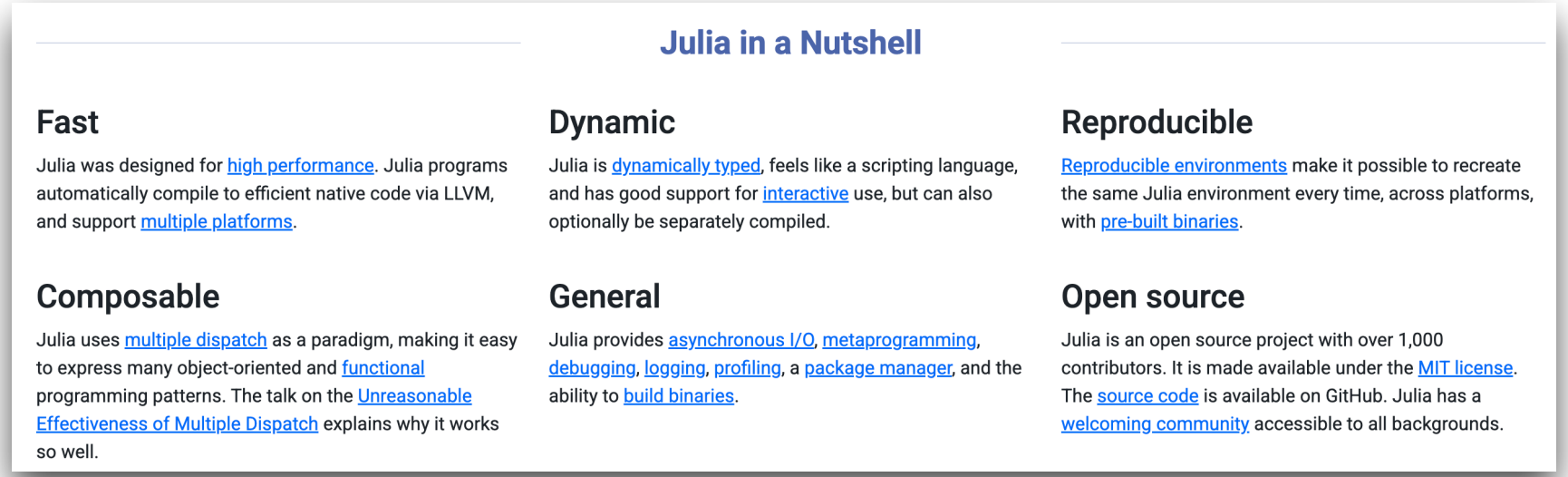
```python
import cProfile
import pstats
from functools import wraps

import os
import time
import numpy as np
from driver28b import Driver28b  # Assuming Driver28b is a module
from driver28c import Driver28c  # Assuming Driver28c is a module

#Wrap Driver28b and Driver28c in a decorator to profile the functions
def profile(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        datafn = func.__name__ + ".profile" # Name the data file sensibly
        prof = cProfile.Profile()
        retval = prof.runcall(func, *args, **kwargs)
        prof.dump_stats(datafn)
        return retval
    return wrapper

# PARAMETERS FOR THE SELECTED EXERCISES (DO NOT CHANGE)
x0, y0 = 0, 0
L1, L2 = 1, 1
noelms1, noelms2 = 40, 50
lam1, lam2 = 1, 1
fun = lambda x, y: np.cos(np.pi * x) * np.cos(np.pi * y)
qt = lambda x, y: 2 * np.pi**2 * np.cos(np.pi * x) * np.cos(np.pi * y)

Driver28b = profile(Driver28b)
Driver28c = profile(Driver28c)

p2b = pstats.Stats('Driver28b.profile')
p2b.sort_stats('cumulative').print_stats(10)

p2c = pstats.Stats('Driver28c.profile')
p2c.sort_stats('cumulative').print_stats(10)

print(p2b)
print(p2c)
```

Sparse computations in Julia

Julia

Installing Julia

- https://julialang.org/downloads/
- https://www.julia-vscode.org/
- https://julialang.org/#editors



Julia in a Nutshell

Fast

Julia was designed for high performance. Julia programs automatically compile to efficient native code via LLVM, and support multiple platforms.

Dynamic

Julia is dynamically typed, feels like a scripting language, and has good support for interactive use, but can also optionally be separately compiled.

Reproducible

Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable

Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General

Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and the ability to build binaries.

Open source

Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub. Julia has a welcoming community accessible to all backgrounds.

Dense Matrix Functions

Julia modules: **LinearAlgebra**, **SparseArrays**, **Random**

- Elementary matrices: **I, zeros, ones, rand, randn, diagm, ...**
- Basis array information: **size, length, ndims, eltype, ...**
- Working with matrices: **reshape, repeat, hcat, vcat, ...**
- Reordering algorithms: **reverse, permutedims, shuffle, sortperm, ...**
- Linear algebra: **eigen, svd, lu, qr, cholesky, cond, norm, ...**
- Linear equations (iterative methods): **cg, gmres, bicgstabl, lsqr, ... (from IterativeSolvers.jl)**
- Specialized matrices: **Symmetric, Hermitian, Diagonal, Tridiagonal, UpperTriangular, LowerTriangular, ...**

Sparse Matrix Functions

Julia modules: **SparseArrays**, **LinearAlgebra**

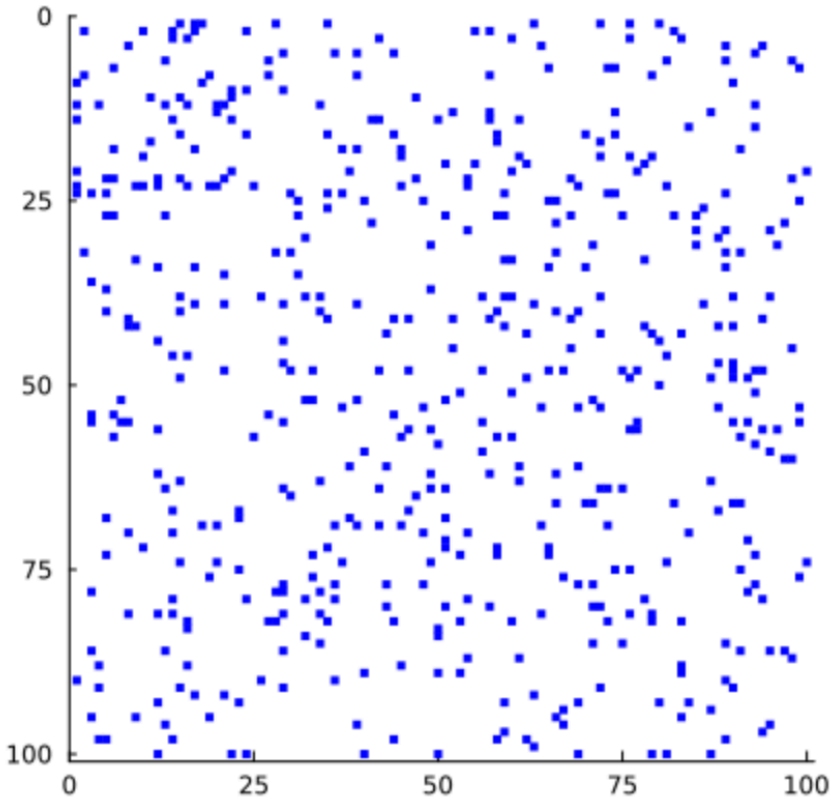
- Elementary sparse matrices: **spzeros, sparse, sprand, sprandn, spdiagm**, ...
- Full to sparse conversion: **sparse, SparseMatrixCSC, findnz, dropzeros!, ...**
- Working with sparse matrices: **nnz, issparse, nonzeros, nzrange, rowvals, spy, ...**
- Reordering algorithms: **reverse, permute!, ...** (also see AMD.jl, Metis.jl for graph ordering)
- Linear algebra: **eigen (for dense), svds, cond, norm, lu, qr, cholesky, ...** (sparse-aware)
- Linear equations (iterative methods): **cg, cgs, gmres, bicgstabl, lsmr, ...** (from IterativeSolvers.jl)
- Miscellaneous: **ldlt, lufact, ** (backslash operator for solve), ...

Sparse matrices

Example:

A 100x100 matrix with 10.000 elements and few nonzeros.

```
using SparseArrays, Plots

# sprand(rows, cols, density)
sp = sprand(100, 100, 0.05)

# Create a 100x100 sparse matrix with approximately 5% non-zero entries
print(nnz(sp))

# Get the row and column indices of nonzero elements
rows, cols, vals = findnz(sp)
scatter(cols, rows,
    markersize=2,
    markercolor=:blue,
    markershape=:square,
    markerstrokewidth=0,
    legend=false,
    yflip=true,
    aspect_ratio=:equal,
    grid=false,
    xlimits=(0, size(sp, 2)+1),
    ylimits=(0, size(sp, 1)+1)
)
```



Data structures for matrices

Conversion between full/Sparse:

```
using SparseArrays
using LinearAlgebra
As = sprand(100, 100, 0.05)  # Sparse matrix
Af = Array(As)  # Dense matrix conversion (or: Matrix(As))
typeof(As) # Output: SparseMatrixCSC{Float64, Int64} with above
```

Julia workspace memory usage:

```
using Base: summarysize
println("Size of dense matrix in bytes: $(summarysize(Af))")
println("Size of sparse matrix in bytes: $(summarysize(As))")
```

Size of dense matrix in bytes: 80128
Size of sparse matrix in bytes: 10112

Comparable to Matlab since using same internal storage format

Extraction of sparse matrix in "coordinate" format:

```
rows, cols, vals = findnz(As) # Note: In Julia, findnz returns (rows, cols, values) tuple
```

Creation of sparse matrix directly:

```
As_direct = sparse(rows, cols, vals)
# or specify also the dimensions if known
As_direct = sparse(rows, cols, vals, 100, 100)
```

Sparse arrays

Compare full/Sparse:

```
using SparseArrays, LinearAlgebra, BenchmarkTools

A = [(i+j)^2 * 100 for i in 1:100, j in 1:100]
I_dense = Matrix{Float64}(I, 100, 100)
I_sparse = sparse(I, 100, 100)

# Benchmark dense
@btime $A * $I_dense;

# Benchmark sparse
@btime $A * $I_sparse;
```

```
4.174 µs (3 allocations: 96.08 KiB)
  3.262 µs
```

```
Dense matrix multiplication took 0.004174 seconds.
Sparse matrix multiplication took 0.003262 seconds...!!
```

Sparse matrix-matrix operation pays off also in Julia

Julia code profiling

drivers.jl

```
using SparseArrays
using LinearAlgebra

"""
TODO: Implement Driver28b
Description: [What this function should do]
"""
function Driver28b()
    # TODO: Student implementation

    return result
end

"""
TODO: Implement Driver28c
Description: [What this function should do]
"""
function Driver28c()
    # TODO: Student implementation

    return result
end

# Export functions
export Driver28b, Driver28c
```

Julia code profiling

main.jl

```
# benchmark_drivers.jl
using BenchmarkTools
include("drivers.jl")

println("Benchmarking Driver28b:")
@btime Driver28b()

println("\nBenchmarking Driver28c:")
@btime Driver28c()

# For detailed profiling
println("\n" * "="^80)
println("Detailed profile of Driver28b:")
@profview Driver28b()  # Opens visual profiler if ProfileView.jl is installed
```

Solution of linear systems of equations

Problem solving

Model problem (see (1.14) and (1.15) in FEM notes)

$$u'' - u = 0, \quad 0 \leq x \leq L$$

$$u(0) = c, \quad u(L) = d$$

Discretization using FEM based on weak formulation (see (1.19) in notes)

$$\int_0^L (u'v' + uv)\, dx = 0$$

Assume representation of solution in the form

$$\hat{u}(x) = \sum_{j=1}^{M} \hat{u}_j N_j(x)$$

Ax=b

Definition of A and b

By Galerkin Method we can generate coupled set of equations

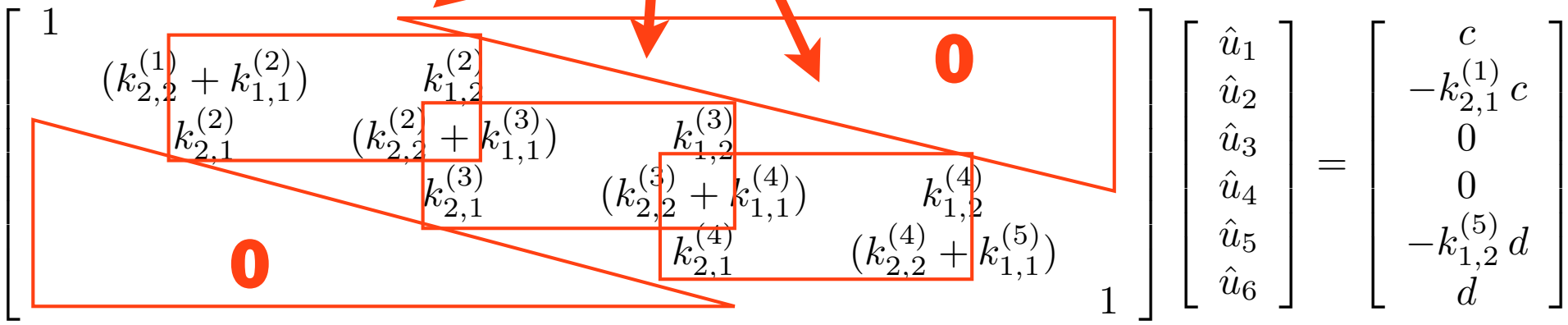
$$\sum_{j=1}^{M} a_{i,j}\hat{u}_j = 0, \quad i = 2, 3, ..., M-1$$

$$a_{i,j} = \int_0^L (N_i' N_j' + N_i N_j)\, dx$$

System of equations

Local to global assembly via $\mathcal{K}^{(i)} = \begin{bmatrix} k_{1,1}^{(i)} & k_{1,2}^{(i)} \\ k_{2,1}^{(i)} & k_{2,2}^{(i)} \end{bmatrix}$

Sparse system!



Compact representation: $\mathbf{A}\hat{\mathbf{u}} = \mathbf{b}$

Does a solution exist? How to solve systems?

Existence and uniqueness

We consider solutions to

$$\mathrm{A}\mathbf{x} = \mathbf{b}$$

Where $\mathrm{A} \in \mathbb{R}^{n \times n}$ is a regular matrix and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ are real-valued.

It can be shown that when A is nonsingular then the inverse matrix A^{-1} exist and the unique solution to the system is

$$\mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$$

If A is singular, it can be shown that the number of solutions is determined by the right hand side vector b. (not considered in this context)

VALIDATION HINT: For small systems (n small) a dirty (albeit in general expensive) way to test for existence and uniqueness of a regular system is to try and compute

```
>> inv(A); % Requires complete assembly of A
```

and see if Matlab detects A being singular and gives a warning.

Assembly of sparse coefficient matrix

Algorithm

Allocate storage for $\mathbf{A}$ and $\mathbf{b}$

for $i := 1$ **to** $M - 1$

Compute $k_{1,1}^{(i)}$, $k_{1,2}^{(i)}$, $k_{2,1}^{(i)}$ and $k_{2,2}^{(i)}$ from (1.26).

$a[i,i] := a[i,i] + k_{1,1}^{(i)}$

$a[i,i+1] := k_{1,2}^{(i)}$

$a[i+1,i] := k_{2,1}^{(i)}$

$a[i+1,i+1] := k_{2,2}^{(i)}$

Update $\mathbf{A}$ and $\mathbf{b}$ to impose boundary conditions

End of algorithm

Preallocate sparse A

Assemble in sparse format (i,j,s)
Then A = sparse(i,j,s);

Note:

- A global assembly of the coefficient matrix (cf. Algorithm 1) is typically required for a direct solution method. Assembly directly in sparse format most efficient for large A.
- For iterative methods, often A is not formed explicitly (matrix-free method)

Assembly of sparse coefficient matrix

Matlab:

```
function [A,b] = GlobalAssembly(M,h)
% Algorithm
nnzmax = 4*M;
ii = ones(nnzmax,1);
jj = ones(nnzmax,1);
ss = zeros(nnzmax,1);
b  = zeros(M,1);
count = 0;
for i = 1 : M-1
    ii(count + (1:4)) = [i i   i+1 i+1];
    jj(count + (1:4)) = [i i+1 i+1 i  ];
    ss(count + (1:4)) = [ElementMatrixElement(1,1,i,h) ...
        ElementMatrixElement(1,2,i,h) ElementMatrixElement(2,2,i,h) ...
        ElementMatrixElement(2,1,i,h) ];
    count = count + 4;
end
A = sparse(ii(1:count),jj(1:count),ss(1:count));
return
```

Note: Boundary conditions not taken into account here (cf. Algorithm 2).

Assembly of sparse coefficient matrix

Python:

```
import numpy as np
from scipy.sparse import csr_matrix

def GlobalAssembly(M, h):
    nnzmax = 4 * M
    ii = np.ones(nnzmax, dtype=int)
    jj = np.ones(nnzmax, dtype=int)
    ss = np.zeros(nnzmax)
    b = np.zeros(M)
    count = 0

    for i in range(M - 1):
        ii[count:count + 4] = [i, i, i + 1, i + 1]
        jj[count:count + 4] = [i, i + 1, i + 1, i]
        ss[count:count + 4] = [
            ElementMatrixElement(1, 1, i, h),
            ElementMatrixElement(1, 2, i, h),
            ElementMatrixElement(2, 2, i, h),
            ElementMatrixElement(2, 1, i, h)
        ]
        count += 4

    A = csr_matrix((ss[:count], (ii[:count], jj[:count])), shape=(M, M))
    return A, b
```

Note: Boundary conditions not taken into account here (cf. Algorithm 2).

Direct Methods

Tridiagonal system (simple):

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= c_1 \\ u_{22}x_2 + \cdots + u_{2n}x_n &= c_2 \\ &\vdots \\ u_{nn}x_n &= c_n \end{aligned} \quad \Rightarrow \quad U = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ & & u_{nn} \end{bmatrix}$$

Solution by backward substitution (elimination).

$$x_n = c_n / u_{nn}$$

$$x_i = \left(c_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii} \ , \quad i = n-1, n-2, \ldots, 1 \ .$$

Idea of Gaussian Elimination methods: Problem Transformation

Rewrite a system into simple tridiagonal form and solve by backward or forward elimination.

Gaussian Elimination

To find a direct solution to the system Ax=b it is possible to use a Gaussian Elimination procedure.

Assume $$A = \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{c} & B \end{bmatrix}$$

First step is to eliminate the nonzero elements below the diagonal element of the first row of A by scaled row eliminations

$$\hat{A} = \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{0} & C \end{bmatrix}, \quad C = B - \frac{\mathbf{c}\mathbf{b}^T}{a}$$

After first step, we have found a factorization

$$A = \begin{bmatrix} 1 & \mathbf{0} \\ \frac{\mathbf{c}}{a} & I \end{bmatrix} \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{0} & C \end{bmatrix}$$

Repeat procedure on the new sub-matrix C to ultimately end up with A=LU, where L is a unit lower triangular matrix and U an upper triangular matrix.

The final solution x is obtained by solving at most two triangular systems.

Direct Methods

Regular system (general):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= c_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= c_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= c_n \end{aligned} \quad \Rightarrow \quad A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

Factorization (rewrite to simple triangular forms using elimination procedure)

$$\boxed{A = LU}$$

$$\begin{aligned} Ax &= b \\ \Rightarrow \quad LUx &= b \\ \Rightarrow \quad Lz = b, &\quad Ux = z \end{aligned}$$

$$L = \begin{bmatrix} l_{11} & & \\ \vdots & \ddots & \\ l_{n1} & \cdots & l_{nn} \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ & & a_{nn} \end{bmatrix}$$

Solution in two steps using backward and forward substitution.

Direct Method for a system with symmetry

System with structure (symmetry):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= c_1 \\ a_{12}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= c_2 \\ &\vdots \\ a_{1n}x_1 + a_{2n}x_2 + \cdots + a_{nn}x_n &= c_n \end{aligned} \quad \Rightarrow A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{bmatrix}$$

Note: If A is symmetric, we only need to store upper or lower triangular elements.

Cholesky factorization (express factorization in terms of triangular matrix)

$$\boxed{\begin{aligned} &\text{Positive Definite} \\ &xAx^T > 0 \\ &A = C^T C \end{aligned}} \Rightarrow \begin{aligned} Ax &= b \\ C^T Cx &= b \\ C^T z &= b, \quad Cx = z \end{aligned} \qquad C = \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ & \ddots & \vdots \\ & & c_{nn} \end{bmatrix}$$

Solution in two steps using backward and forward substitution with only one storage location (C is about half the size of A).

Direct Methods in Matlab

```
>> A = [4 6 2; 6 18 -1.5; 2 -1.5 4.25];
>> C = chol(A)
C = 2.0000    3.0000    1.0000
         0    3.0000   -1.5000
         0         0    1.0000
```

$$\mathbf{A} = \begin{bmatrix} 4 & 6 & 2 \\ 6 & 18 & -1.5 \\ 2 & -1.5 & 4.25 \end{bmatrix}$$

```
>> A(3,3) = 3.25;  C1 = chol(A)
??? Error using ==> chol
Matrix must be positive definite.
```

$$\mathbf{A} = \begin{bmatrix} 4 & 6 & 2 \\ 6 & 18 & -1.5 \\ 2 & -1.5 & 3.25 \end{bmatrix}$$

```
>> [C2, p] = chol(A)
C2 = 2     3
     0     3
p = 3
```

Direct Methods in Python

```python
import numpy as np
from scipy.linalg import cholesky, LinAlgError

# Define the matrix A
A = np.array([[4, 6, 2], [6, 18, -1.5], [2, -1.5, 4.25]])

# Attempt Cholesky decomposition
try:
    C = cholesky(A, lower=False)
    print("Cholesky decomposition successful. Matrix C is:")
    print(C)
except LinAlgError:
    print("Matrix must be positive definite for Cholesky decomposition.")

# Modify the matrix A to make it non-positive definite
A[2, 2] = 3.25

# Attempt Cholesky decomposition again
try:
    C1 = cholesky(A, lower=False)
    print("Cholesky decomposition successful. Matrix C1 is:")
    print(C1)
except LinAlgError:
    print("Matrix must be positive definite for Cholesky decomposition.")

# Use the `cho_factor` function which returns a flag for positive definiteness
from scipy.linalg import cho_factor, cho_solve

try:
    C2, lower = cho_factor(A)
    print("Cholesky decomposition with cho_factor successful.")
except LinAlgError as e:
    print(e)
```

$$\mathbf{A} = \begin{bmatrix} 4 & 6 & 2 \\ 6 & 18 & -1.5 \\ 2 & -1.5 & 4.25 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 4 & 6 & 2 \\ 6 & 18 & -1.5 \\ 2 & -1.5 & 3.25 \end{bmatrix}$$

```
Cholesky decomposition successful. Matrix C is:
[[ 2.  3.  1.]
 [ 0.  3. -1.5]
 [ 0.  0.  1.]]
Matrix must be positive definite for Cholesky decomposition.
3-th leading minor of the array is not positive definite
```

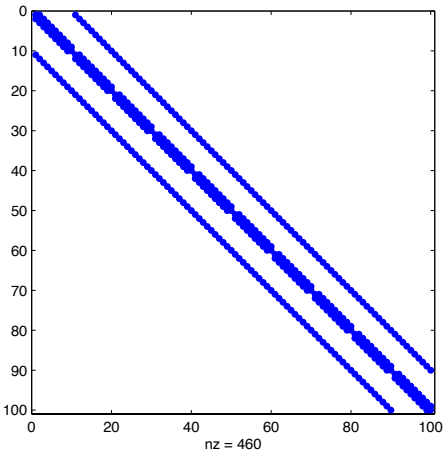
First try

Second try

Sparse band matrix

An important special type of matrix is a band matrix where the nonzero elements are all concentrated close to the main diagonal.

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

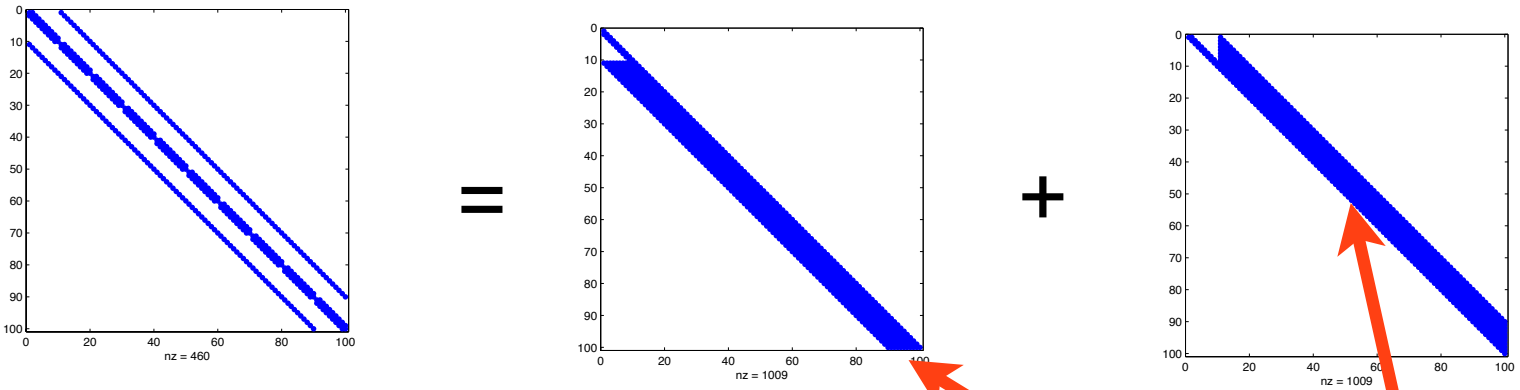


Gaussian elimination for band matrices are similar to the general case, with some minor algorithmic changes for exploiting the special structure. This can lead to great improvements reducing memory requirements and efficiency when the system is large and very sparse.

Note: bandwidth may change with node ordering (see notes pages 45-46)

Sparse Direct Methods

What can happen when zero elements are present in coefficient matrix during the Gaussian Elimination procedure?



A = L + U

Factorization (rewrite to simple triangular forms)

```
>> [L,U]=lu(A);
```

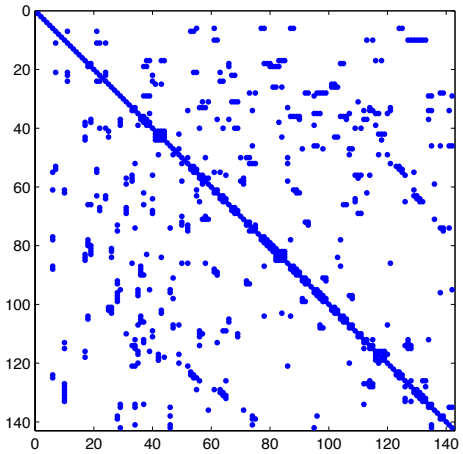
```
>> spy(A)
```

Fill-in can be checked with

```
>> [nnz(A), nnz(L), nnz(U)]

ans =

   460   1009   1009
```

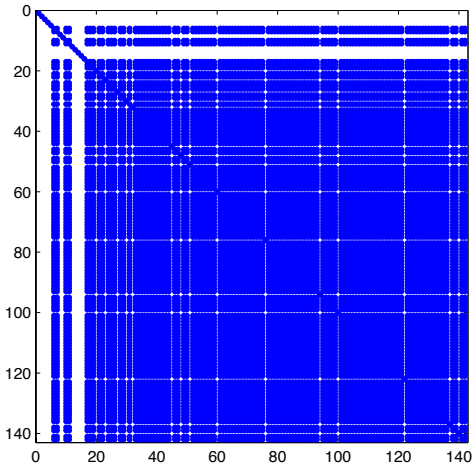
Fill-in between diagonal bands result in increased storage

Sparse Direct Methods




```
>> spy(A)

>> nnz(A)

   ans=766

>> tic; u=A\b; toc

Elapsed time is 0.000849 seconds.
```

Typical worst case:



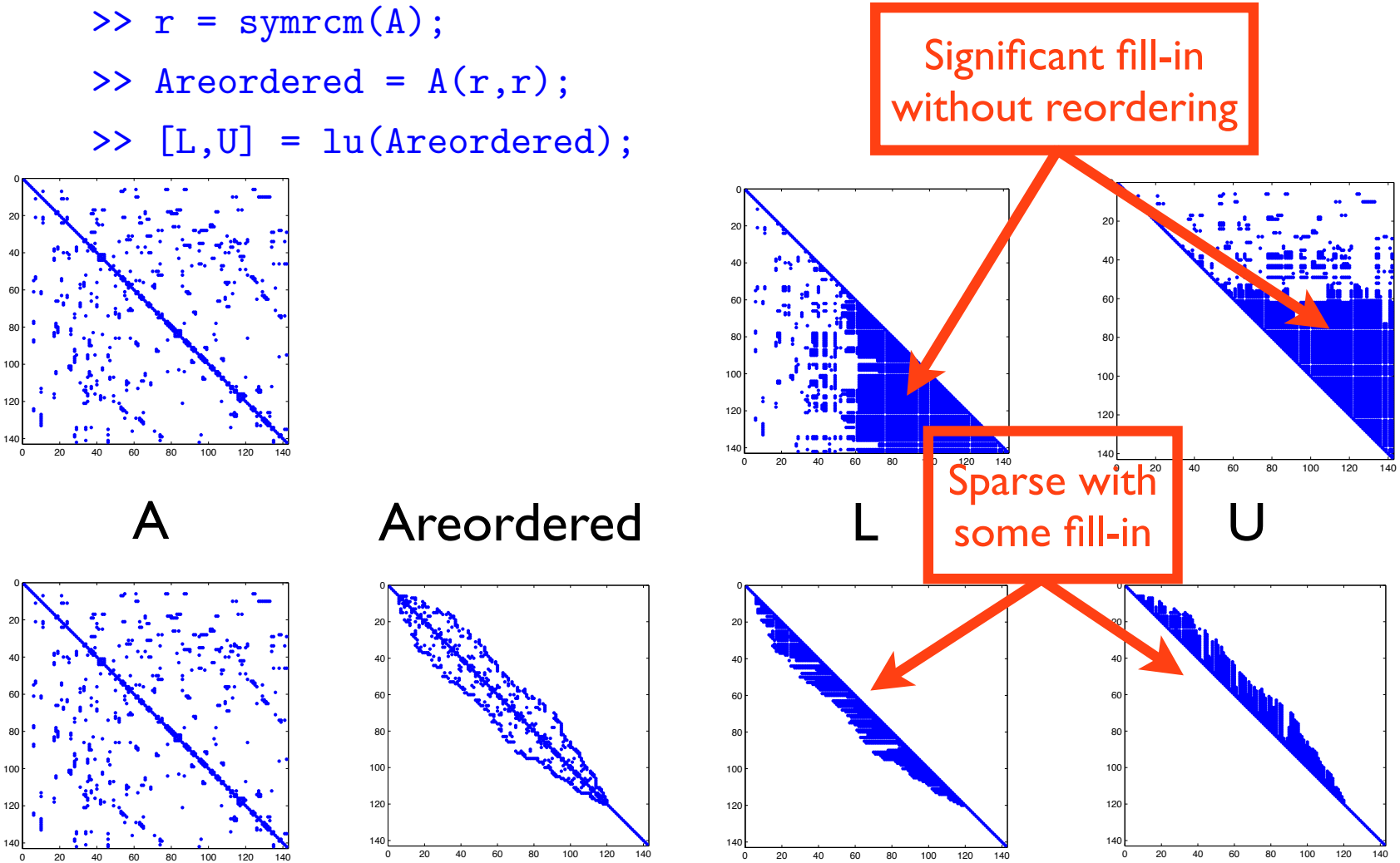

```
>> spy(inv(A))

>> nnz(inv(A))

      ans=13252

>> tic; u=inv(A)*b; toc

Elapsed time is 0.044910 seconds.
```

Never do this in practice!

Sparse Direct Methods

Permutation (reordering) before factorization can reduce fill-in (significantly) since the amount is sensitive to the order in which rows and columns are processed

```
>> r = symrcm(A);
>> Areordered = A(r,r);
>> [L,U] = lu(Areordered);
```



Fill-in minimizing reordering algorithms: symmmd, symamd, symrcm, amd, colamd,...

Sparse Direct Methods

Sparse Direct methods can improve efficiency (over dense methods) and memory footprint by exploiting sparse structure. Examples of useful commands

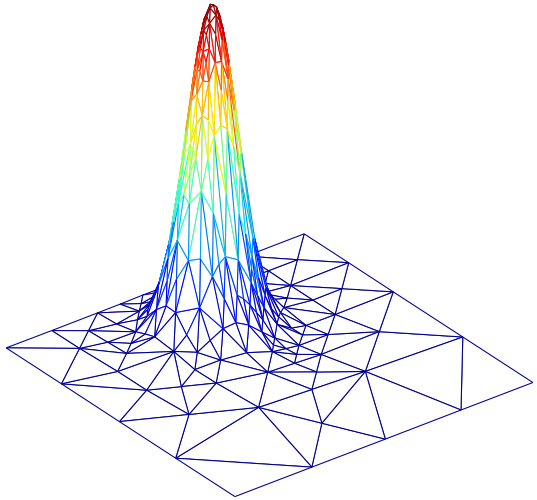
```
>> r = symrcm(A);
>> Areordered = A(r,r);
>> [L,U] = lu(Areordered);
```

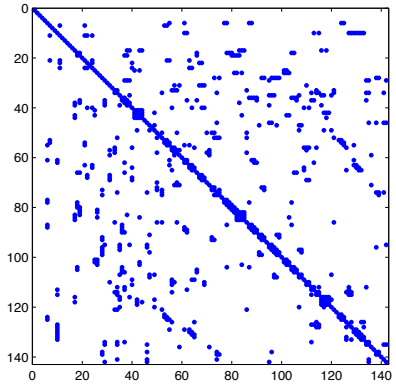
```
>> tic; u=A\b; toc
Elapsed time is 0.000748 seconds.
```

```
>> A=full(A); tic; u=A\b; toc
Elapsed time is 0.343455 seconds.
```

```
>> Af = full(A);
>> As = sparse(A);
>> whos Af As
```

Name	Size	Bytes	Class	Attributes
Af	142x142	161312	double	
As	142x142	13400	double	sparse





Direct Methods in Matlab

In Matlab solving linear systems is straightforward

```
>> x = A \ b;
```

Within this "black box" different checks (triangularity, needs for permutations, banded matrix, etc) and algorithms (direct/sparse) are hidden for the user.

Instead, choosing the solver (to reduce overhead) can be done

```
>> help linsolve
```

To define parameters for sparse matrix routines in Matlab

```
>> help spparms
```

Do **NOT(!)** solve using inverse matrices as

```
>> x = inv(A)*b;
```

Since this usually requires **much(!)** more computational resources than Gaussian Elimination procedures.

Direct Methods in Python

In Python solving linear systems requires selecting packages, cf.

```python
import numpy as np
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import spsolve

# Example dense matrix and vector
A_dense = np.array([[3, 2, -1], [2, -2, 4], [-1, 0.5, -1]])
b_dense = np.array([1, -2, 0])

# Solve the dense system
x_dense = np.linalg.solve(A_dense, b_dense)

# Example sparse matrix and vector
A_sparse = csc_matrix(A_dense)
b_sparse = b_dense

# Solve the sparse system
x_sparse = spsolve(A_sparse, b_sparse)

# Print the results
print("Solution for dense system:", x_dense)
print("Solution for sparse system:", x_sparse)
```

What's behind '\' in Matlab?

```
>> x = A \ b;
```

In Matlab, the '\' command invokes an algorithm which depends upon the structure of the matrix A.

1. If A is sparse and banded, employ a banded solver.
2. If A is an upper or lower triangular matrix, employ a backward substitution algorithm.
3. If A is symmetric and has real positive diagonal elements, attempt a Cholesky factorization. If A is sparse, employ reordering first to minimize fill-in.
4. If none of criteria above is fulfilled, do a general triangular factorization using Gaussian elimination with partial pivoting.
5. If A is sparse, then employ the UMFPACK library.
6. If A is not square, employ algorithms based on QR factorization for undetermined systems.

Note: detecting appropriate algorithm results in some overhead.

If properties are known, then **linsolve** can be used.

Scaling of effort

Estimates for the computational cost can be expressed in terms of required number of floating point operations (flops) to successfully complete the algorithms, e.g.

For a system $Ax = b$ with regular matrix $A \in \mathbb{R}^{n \times n}$ it can be shown that the asymptotic (i.e. large n) cost of factorization and solve steps are

Structure of A	**Factorization**	**Solve**
Dense, General	(2/3)n^3	2n^2
Dense, Symmetric and Positive Definite	(1/3)n^3	2n^2
Triangular	0	n^2
Sparse k-banded, General	4nk^2	6nk
Sparse k-banded, Symmetric	2nk^2	4nk

O(n) scaling

The estimates are useful to predict computational times and scalability properties since compute time can be assumed to be proportional to flop count.

Problem: work and storage requirements for large problems (large n) may be prohibitive for direct methods (mainly due to fill-in). If sparse techniques can be exploited, the penalties can be (much) less severe.

Iterative methods

We want to solve

$$Au = b$$

Ideas:
- A solution need not be accurate to finite precision
  (errors are committed anyways in the discrete solution process)
- Compute successive approximations $u^{[0]}, u^{[1]}, \ldots$

Main requirements
- a useful partial result is achieved with small effort (few iterations)
- a stop or convergence criterion to achieve desired acceptable error level
- solution effort should be less than direct methods for same problem
- it should be possible to compute the matrix-vector product Au efficiently
- economical with respect to storage
  (usually no fill-in at all and avoid forming A (matrix-free))
- easy to program (in Matlab several standard iterative solvers are available)
- a known approximate solution can be exploited
  (initial guess to reduce iteration count)

Error definitions

For the evaluation and understanding of performance of iterative methods, there are three errors we need to pay attention to

- Discretization error: Difference between exact solution of model and exact solution of discrete equations
  $(u - u_h)$
- Algebraic error: Difference between exact solution to discrete equations and approximate solution
  $(u_h - \hat{u})$
- Total error: Difference between an approximate solution and the exact solution to the model equations.
  $(u - \hat{u})$

Control via mesh parameters

Control via convergence criterium for iterative methods

By the triangle inequality we can bound the total errors

$$||u - \hat{u}|| \leq ||u - u_h + u_h - \hat{u}|| \leq ||u - u_h|| + ||u_h - \hat{u}||$$

For iterative methods, the goal is typically to ensure that the algebraic errors is smaller than the discretization errors.

Stop and convergence criterions

For the solution of Au=b, a stopping criterium for the iterative solvers can be based on measuring the residual errors d=b-Au^[k], e.g. taking the 2-norm of defects

$$||\mathbf{d}||_2 = \left(\mathbf{d}^T \cdot \mathbf{d}\right)^{\frac{1}{2}}$$

For achieving a desired accuracy level, we can use the absolute residual error (atol) or a relative residual error (rtol) as convergence criterions

$$||\mathbf{d}||_2 \leq \texttt{atol} \qquad\qquad ||\mathbf{d}||_2 \leq \texttt{rtol} \cdot ||\mathbf{b}||_2$$

or combination hereof

$$||\mathbf{d}||_2 \leq \texttt{atol} + \texttt{rtol} \cdot ||\mathbf{b}||_2$$

for stopping the iterations.

A criterion of this kind is usually combined with defining a maximum number of iterations (maxiter) to make sure that the iterative solver is stopped before too much effort is wasted (in case of divergence or very slow convergence).

Iterative methods in practice

To solve Au=b it is customary to select (or at least try!) a standard solver

Unsymmetric coefficient matrix
GMRES, BICGSTAB

Symmetric and Positive Definite
Conjugate Gradient (CG)

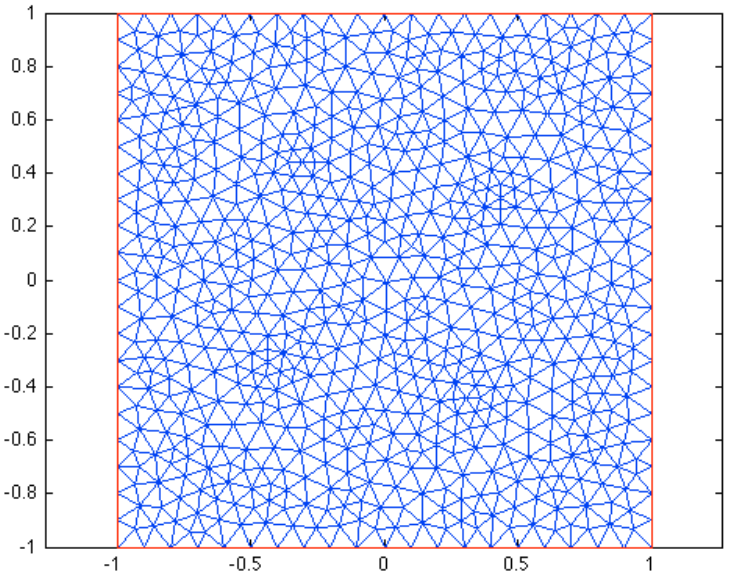
Preconditioning (cheap solve for improving convergence)
LU factorization, Incomplete LU factorization, SParse Approximate Inverse (SPAI), CHOL

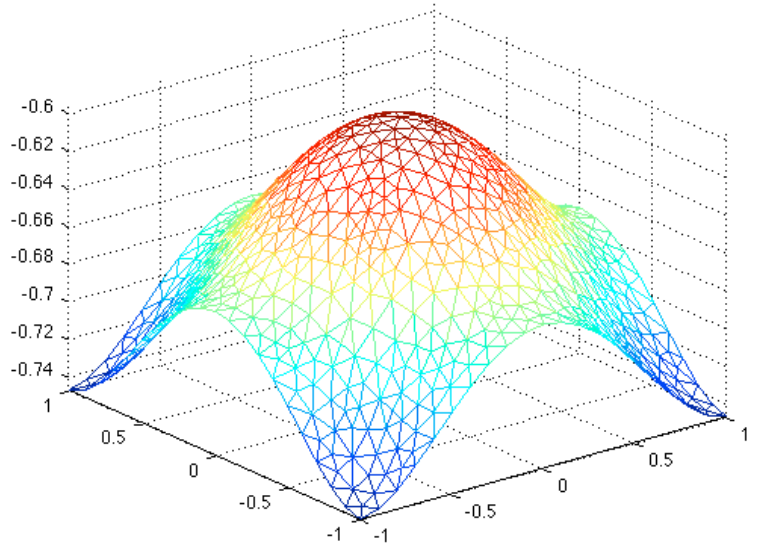
Advanced solvers (typical for really large systems)
Multigrid, Domain Decomposition Methods

NOTE: Easy to both proto-type, test and get experiences in Matlab. Use tic/toc commands to check whether choice was efficient.

Iterative methods in Matlab

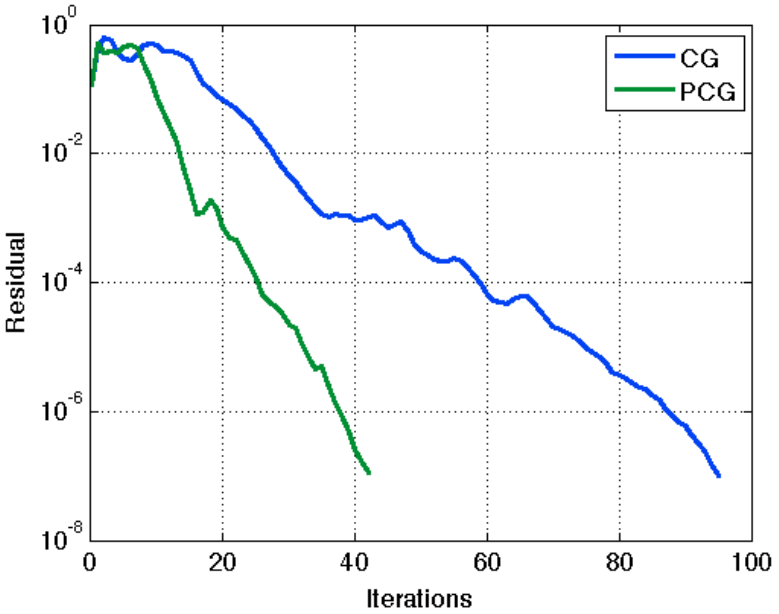
```
% Uses Matlab PDE toolbox
Hmax = 1.0E-01; % maximum edge size
[p,e,t]=initmesh('squareg', 'Hmax', Hmax);
pdemesh(p,e,t), axis equal;
%
% assema assembles the matrices for
%   -div[c grad(u)]+au = f
%
c=1; a=1;
% define right hand side
f = '-(x.^2+y.^2)';
[K,M,b]=assema(p,t,c,a,f);
% global assembly
A=K+M;
% try e.g. spy(A), lu(A), chol(A), etc
u = b*0;
u(:)=A\b(:); % direct solve
% visualize
trimesh(t(1:3,:)',p(1,:)',p(2,:)',u)
```





Iterative methods in Matlab

```
% Solve using conjugate gradient
Ri=cholinc(A,'0');
%b=rand(size(A,1),1);
u_direct = A\b;
% no preconditioning
[u_cg,flag,relres,iter,resvec] = ...
   pcg(A,b,1.0E-06,100);
norm(u_direct-u_cg)
% with cholesky factor for preconditioning
[u_pcg,flag2,relres2,iter2,resvec2] = ...
  pcg(A,b,1.0E-06,100,Ri',Ri);
% algebraic error
norm(u_direct-u_pcg)
```




```
figure
set(gca,'fontsize',14)
semilogy(0:iter,resvec,0:iter2,resvec2,'linewidth',2)
xlabel('Iterations')
ylabel('Residual')
grid on
legend('CG','PCG')
```

Parallel Computing in Matlab

Parallel Computing Toolbox

https://se.mathworks.com/help/parallel-computing/index.html?s_tid=CRUX_lftnav

Parallel Computing Toolbox™ lets you solve computationally and data-intensive problems using multicore processors, GPUs, and computer clusters. High-level constructs—parallel for-loops, special array types, and parallelized numerical algorithms—enable you to parallelize MATLAB® applications without CUDA or MPI programming. The toolbox lets you use parallel-enabled functions in MATLAB and other toolboxes. You can use the toolbox with Simulink® to run multiple simulations of a model in parallel. Programs and models can run in both interactive and batch modes.

The toolbox lets you use the full processing power of multicore desktops by executing applications on workers (MATLAB computational engines) that run locally. Without changing the code, you can run the same applications on clusters or clouds (using MATLAB Parallel Server™). You can also use the toolbox with MATLAB Parallel Server to execute matrix calculations that are too large to fit into the memory of a single machine.

```
% --------
% Running in serial
tic; a1 = ex_serial(50,10000); t1 = toc;

% Running in parallel
gcp; % open parallel pool if none is open
tic; a2 = ex_parallel(50,10000); t2 = toc;

% Compare processing times
disp(['Serial processing time:   ' num2str(t1)])
disp(['Parallel processing time: ' num2str(t2)])

% Comparing results
subplot(1,2,1)
hist(a1, 23:0.2:27), xlim([23 27]), title('serial')
subplot(1,2,2)
hist(a2, 23:0.2:27), xlim([23 27]), title('parallel')

% Copyright 2010 - 2014 The MathWorks, In
```

```
function a = ex_serial(M, N)
% --------
% EX_SERIAL performs N trials of
%  computing the largest eigenvalue
% for an M-by-M random matrix
%
% Inputs:
% M     number of rows and columns
%       of each matrix
% N     number of trials
%
% Output:
% a     vector of largest eigenvalues
%
% Example:
% >> a = ex_serial(50,4000);

a = zeros(N,1);
for I = 1:N
   a(I) = max(eig(rand(M)));
end

% Copyright 2010 - 2014 The MathWorks, Inc.
```

```
function a = ex_parallel(M, N)
% --------
% EX_PARALLEL performs N trials of
%  computing the largest eigenvalue
% for an M-by-M random matrix
%
% Inputs:
% M     number of rows and columns
%       of each matrix
% N     number of trials
%
% Output:
% a     vector of largest eigenvalues
%
% Example:
% >> a = ex_parallel(50,4000);

a = zeros(N,1);
parfor I = 1:N
   a(I) = max(eig(rand(M)));
end

% Copyright 2010 - 2014 The MathWorks, Inc.
```

```
>> ex_compare
Starting parallel pool (parpool) using the 'local' profile ...
Serial processing time:   4.0873
Parallel processing time: 1.0128
IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
```

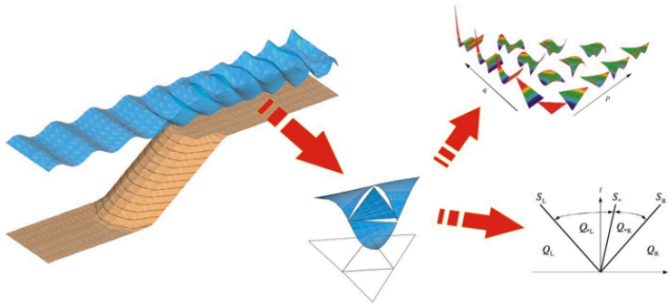
Ongoing work: Research highlights

- new or improved scientific computing methods enable new applications

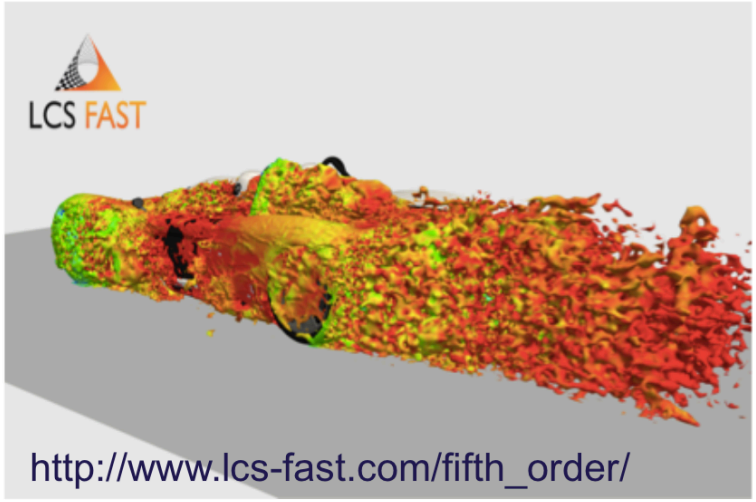
Increased modelling fidelity using high-order numerical methods

SPECTRAL/*hp* ELEMENT METHODS

- *p*-type finite elements due to Babuska et al (1981)
- Spectral element method (SEM) due to Patera (1984)
- Spectral/*hp* Element Methods for CFD (Karniadakis and Sherwin, 1999)
- Nodal high-order DG methods (Hesthaven and Warburton, 2008)
- Today – frameworks (nektar++, nek5000, fenics, deal.II, etc)
- Often used for DNS (uDNS/ILES) simulations of bluff bodies
- Next generation NWP models are high-order DG
- Industrial engineering applications emerging now



NEKTAR++
SPECTRAL/HP ELEMENT FRAMEWORK



✦ High-order methods can be significantly more **cost-efficient** than low-order methods

✦ High-order methods maps to modern **many-core hardware** for improved performance

Increased modelling fidelity using high-order numerical methods

SPECTRAL/*hp* ELEMENT METHODS

- *p*-type finite elements due to Babuska et al (1981)
- Spectral element method (SEM) due to Patera (1984)
- Spectral/*hp* Element Methods for CFD (Karniadakis and Sherwin, 1999)
- Nodal high-order DG methods (Hesthaven and Warburton, 2008)
- Today – frameworks (nektar++, nek5000, fenics, deal.II, etc)
- Often used for DNS (uDNS/ILES) simulations of bluff bodies
- Next generation NWP models are high-order DG
- Industrial engineering applications emerging now

Springer

Available online at https://link.springer.com
http://www.jhydrod.com/
2018,30(1):1-22
https://doi.org/10.1007/s42241-018-0001-1

Spectral/*hp* element methods: Recent developments, applications, and perspectives *

Hui Xu [1], Chris D. Cantwell [1], Carlos Monteserin [2], Claes Eskilsson [4, 5], Allan P. Engsig-Karup [2, 3], Spencer J. Sherwin [1]

1. *Department of Aeronautics, Imperial College London, London SW7 2AZ, UK*
2. *Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark*
3. *Center for Energy Resources Engineering (CERE), Technical University of Denmark, 2800 Kgs. Lyngby, Denmark*
4. *Department of Civil Engineering, Aalborg University, DK-9220 Aalborg Ø, Denmark*
5. *Division Safety and Transport, Research Institutes of Sweden (RISE), SE-50115 Borås, Sweden*

(Received December 22, 2017, Accepted December 28, 2017)

Abstract: The spectral/*hp* element method combines the geometric flexibility of the classical *h*-type finite element technique with the desirable numerical properties of spectral methods, employing high-degree piecewise polynomial basis functions on coarse finite element-type meshes. The spatial approximation is based upon orthogonal polynomials, such as Legendre or Chebychev polynomials, modified to accommodate a C^0- continuous expansion. Computationally and theoretically, by increasing the polynomial order p, high-precision solutions and fast convergence can be obtained and, in particular, under certain regularity assumptions an exponential reduction in approximation error between numerical and exact solutions can be achieved. This method has now been applied in many simulation studies of both fundamental and practical engineering flows. This paper briefly describes the formulation of the spectral/*hp* element method and provides an overview of its application to computational fluid dynamics. In particular, it focuses on the use of the spectral/*hp* element method in transitional flows and ocean engineering. Finally, some of the major challenges to be overcome in order to use the spectral/*hp* element method in more complex science and engineering applications are discussed.

Key words: High-precision spectral/*hp* elements, continuous Galerkin method, discontinuous Galerkin method, implicit large eddy simulation

Project Leader

Spencer Sherwin is Professor of Computational Fluid Mechanics and the Head of Aerodynamics Section in the Department of Aeronautics at Imperial College London. He received his MSE and Ph. D. from the Department of Mechanical and Aerospace Engineering Department at Princeton University in 1995. Prior to this he received his BEng from the Department of Aeronautics at Imperial College London in 1990. In 1995, he joined the Department of Aeronautics at Imperial College as a lecturer and subsequently became a full professor in 2005. Over the past 27 years he has specialised in the development and application of advanced parallel spectral/*hp* element methods for flow around complex geometries with a particular emphasis on vortical and bluff body flows, biomedical modelling of the cardiovascular system and more recently in industrial practice through the partnerships with McLaren Racing and Rolls Royce. Professor Sherwin's research group also develops and distributes the open-source spectral/*hp* element package *Nektar++* (www.nektar.info) which has been applied to direct numerical simulation, large eddy simulation and stability analysis to a range of applications including vortex flows of relevance to offshore engineering and vehicle aerodynamics as well as biomedical flows associated with arterial atherosclerosis. He has published numerous peer-reviewed papers in international journals covering topics from numerical analysis to applied and fundamental fluid mechanics and co-authored a highly cited book on the spectral/*hp* element method. Since 2014 Prof. Sherwin has served as an associate editor of the

* **Biography:** Hui Xu (1981-), Male, Ph. D.
Corresponding author: Hui Xu,
E-mail: hui.xu@imperial.ac.uk

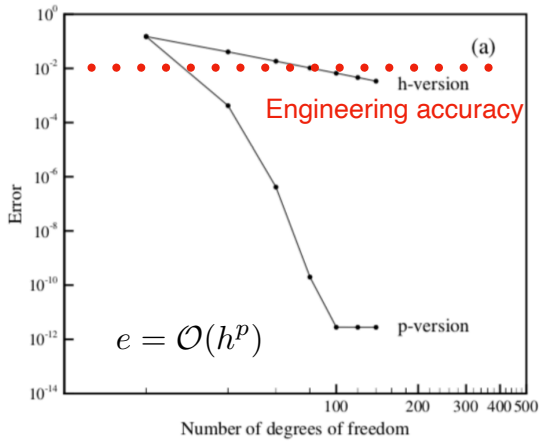
✦ High-order methods can be significantly more **cost-efficient** than low-order methods

✦ High-order methods maps to modern **many-core hardware** for improved performance

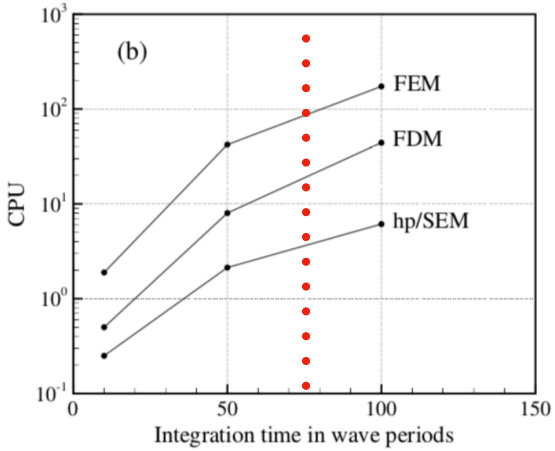
Foundations of unstructured Spectral Element Methods

- for numerical approximation of differential equations

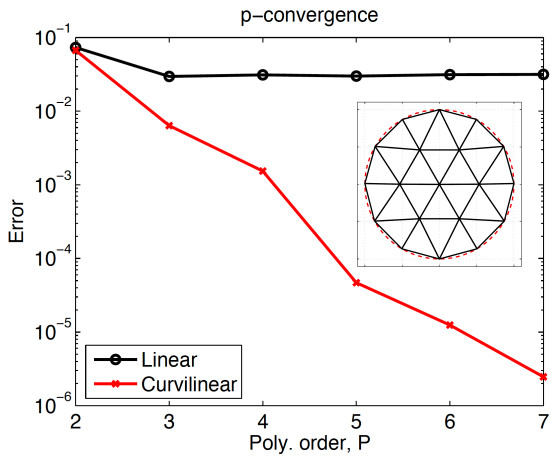
Algorithmic Efficiency



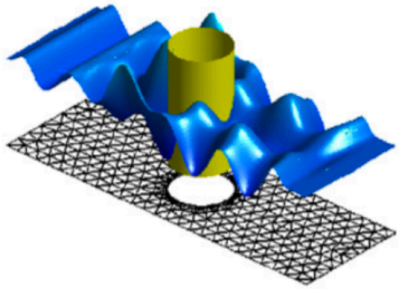
Numerical Efficiency

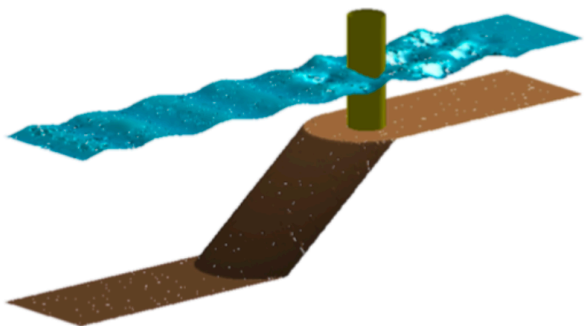


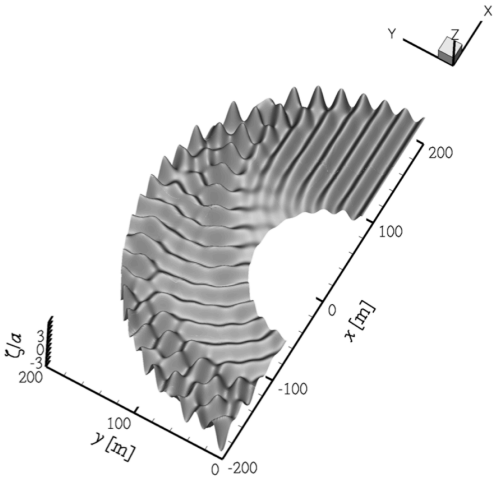
Geometric accuracy



- **Adaptive FEM.** Cost-efficient via h-refinement. Accuracy achieved via adapting mesh to features of problem/solution.
- **Spectral Methods.** Cost-efficient via p-refinement. High accuracy with few degrees of freedom.
- **Accurate Geometric representation.** Cost-efficient via iso-parametric elements.
  High accuracy representation of curved surfaces with few elements.







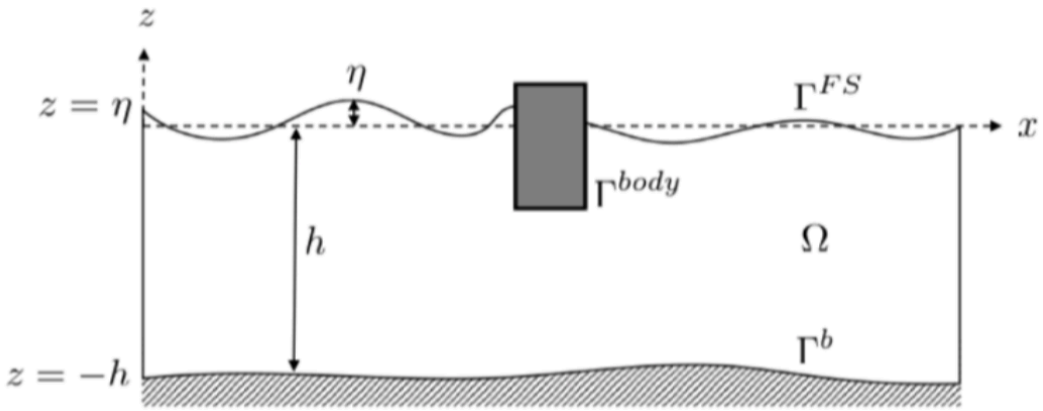
On the need for general-purpose hydrodynamic simulators for offshore sectors






- ✦ Wave-impact loading on offshore marine structures and extreme/rare events.
- ✦ Influence on the bathymetry and marine structures of the sea state in near-coastal regions.
- ✦ Seakeeping performance, wave energy converters, floating production facilities, etc.

Eulerian Fully Nonlinear Potential Flow (FNPF) Equations



Dynamic and kinematic free surface boundary conditions

$$\partial_t \zeta = -\boldsymbol{\nabla}\zeta \cdot \boldsymbol{\nabla}\tilde{\phi} + \tilde{w}(1 + \boldsymbol{\nabla}\zeta \cdot \boldsymbol{\nabla}\zeta),$$
$$\partial_t \tilde{\phi} = -g\zeta - \frac{1}{2}\left(\boldsymbol{\nabla}\tilde{\phi} \cdot \boldsymbol{\nabla}\tilde{\phi} - \tilde{w}^2(1 + \boldsymbol{\nabla}\zeta \cdot \boldsymbol{\nabla}\zeta)\right),$$

Laplace problem (mass conservation)

$$\phi = \tilde{\phi}, \quad z = \zeta(\mathbf{x}, t),$$
$$\boldsymbol{\nabla}^2\phi + \partial_{zz}\phi = 0, \quad -h \leq z < \zeta(\mathbf{x}, t),$$
$$\partial_z \phi + \boldsymbol{\nabla} h \cdot \boldsymbol{\nabla}\phi = 0, \quad z = -h.$$

- Assumptions: non-breaking waves, inviscid irrotational and incompressible fluid flow.
- Free surface equations are nonlinear and time-dependent.

Numerical discretization of an Eulerian FNPF model

Introduce the space of continuous, piece-wise polynomial functions $V = \{v_h \in C^0(\Omega); \forall k \in \{1, ..., N^k\}, v_{h|\mathcal{T}_k} \in \mathbb{P}^P\}$. The weak Galerkin formulation of the free surface boundary conditions takes the following form. Find $f \in V$ where $f = \eta, \tilde{\phi}$ such that

$$\iint_{\Gamma^{FS}} \partial_t \eta v(\mathbf{x}) d\mathbf{x} = \iint_{\Gamma^{FS}} [-\boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\tilde{\phi} + \tilde{w}(1 + \boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\eta)] v(\mathbf{x}) d\mathbf{x}$$

$$\iint_{\Gamma^{FS}} \partial_t \tilde{\phi} v(\mathbf{x}) d\mathbf{x} = \iint_{\Gamma^{FS}} \left[-g\eta - \frac{1}{2}\left(\boldsymbol{\nabla}\tilde{\phi} \cdot \boldsymbol{\nabla}\tilde{\phi} - \tilde{w}^2(1 + \boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\eta) \right) \right] v(\mathbf{x}) d\mathbf{x}$$

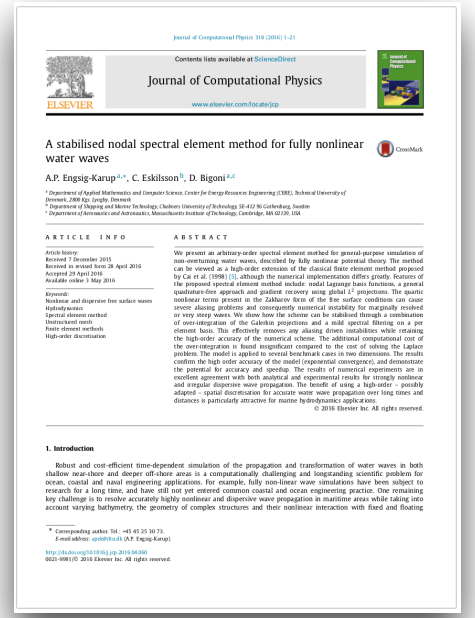
Introduce the finite-dimensional approximations

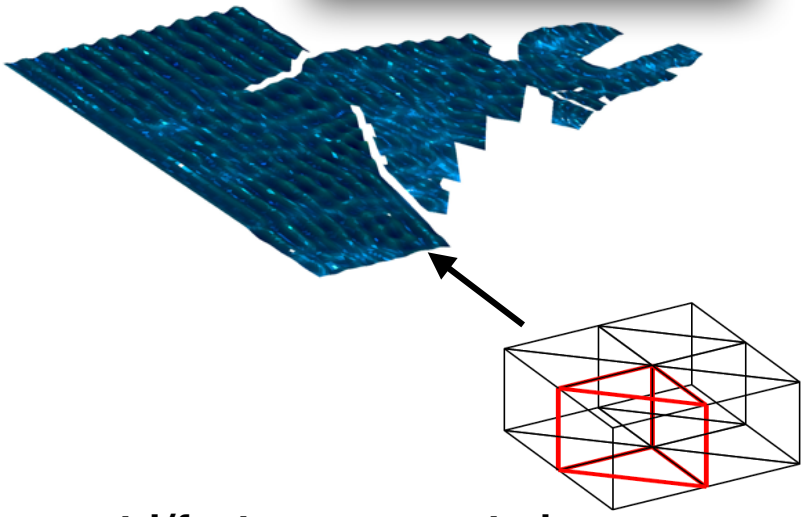
$$f_h(\mathbf{x}, t) = \sum_{j=1}^{N^{FS}} f_j(t) N_j(\mathbf{x}), \quad \iint_\Omega \nabla f_h N_i d\mathbf{x} = \iint_\Omega \sum_{j=1}^{N^{FS}} f_j \nabla N_j N_i d\mathbf{x}$$

where $\{N_i\}_{i=1}^{N_{FS}} \in V$ is the set of global finite element basis functions on the mesh. The weak formulation of the Laplace problem is find $\Phi \in V$ such that

$$\iiint_\Omega \nabla \cdot \nabla\phi v d\mathbf{x} = \oint_\Gamma v\mathbf{n} \cdot \nabla\phi d\mathbf{x} - \iiint_\Omega \nabla\phi \cdot \nabla v d\mathbf{x} = 0$$

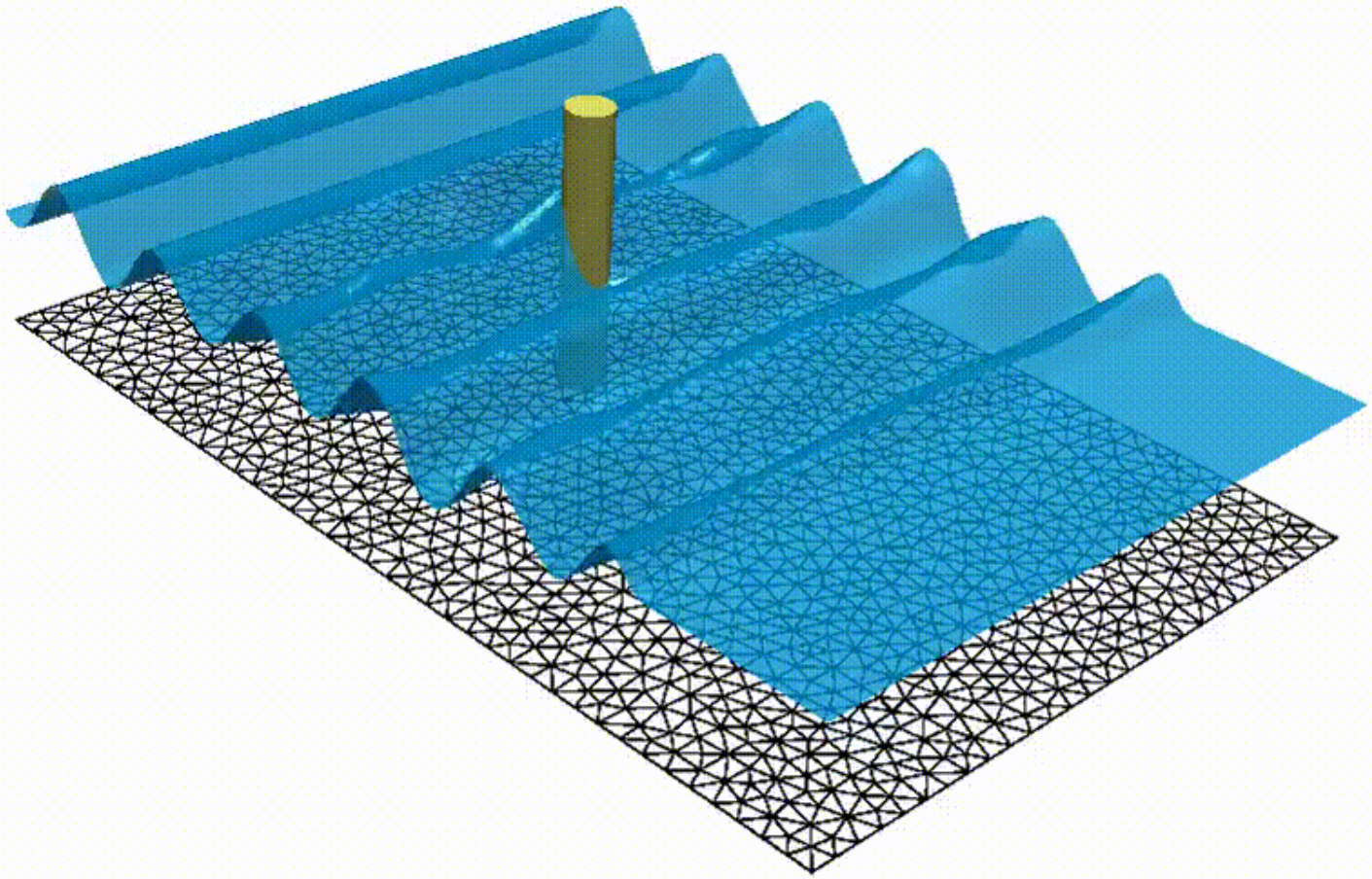
for all $v \in V$.





- ✦ **Multi-element (domain decomposition) adaptive mesh for geometri/features supported.**
- ✦ **Gradient Recovery of derivatives on unstructured meshes using global L2 Projection.**
- ✦ **Temporal Integration via a 4th order Explicit Runge-Kutta method (ERK4).**

Numerical discretization of an Eulerian FNPF model using SEM



Wave-body interaction with a fixed FPSO (3D)

CCP-WSI

A Collaborative Computational Project in Wave Structure Interaction

https://www.ccpwsi.ac.uk/

Proceedings of the Twenty-eighth (2018) International Ocean and Polar Engineering Conference www.isope.org
Sapporo, Japan, June 10-15, 2018

ISBN 978-1-880653-87-6; ISSN 1098-6189

Spectral Element FNPF Simulation of Focused Wave Groups Impacting a Fixed FPSO

Allan P. Engsig-Karup
Department of Applied Mathematics and Computer Science
Center for Energy Resources Engineering (CERE)
Technical University of Denmark
Kongens Lyngby, Denmark

Claes Eskilsson
Research Institutes of Sweden (RISE)
Borås, Sweden
Department of Civil Engineering
Aalborg University
Aalborg Ø, Denmark

ABSTRACT

For the assessment of experimental measurements of focused wave groups impacting a surface-piecing fixed structure, we present a new Fully Nonlinear Potential Flow (FNPF) model for simulation of unsteady water waves. The FNPF model is discretized in three spatial dimensions (3D) using high-order prismatic - possibly curvilinear - elements using a spectral element method (SEM) that has support for adaptive unstructured meshes. This SEM-FNPF model is based on an Eulerian formulation and deviates from past works in that a direct discretization of the Laplace problem is used making it straightforward to handle accurately floating structural bodies of arbitrary shape. Our objectives are; i) present detail of a new SEM modelling developments and ii) to consider its application to address a wave-body interaction problem for nonlinear design waves and their interaction with a model-scale fixed Floating Production, Storage and Offloading vessel (FPSO). We first reproduce experimental measurements for focused design waves that represent a probably extreme wave event for a sea state represented by a wave spectrum and seek to reproduce these measurements in a numerical wave tank. The validated input signal based on measurements is then generated in a NWT setup that includes the FPSO and differences in the signal caused by nonlinear diffraction is reported.

KEY WORDS: Spectral element method; high order numerical methods; unstructured meshes; fully nonlinear potential flow; focused wave; wave-body interaction; FPSO.

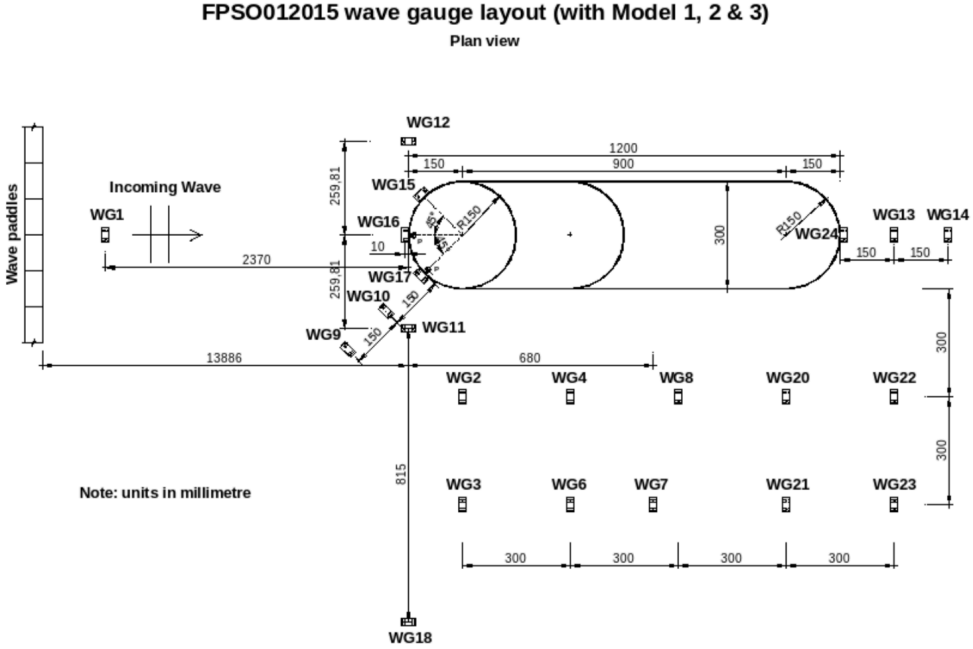
INTRODUCTION

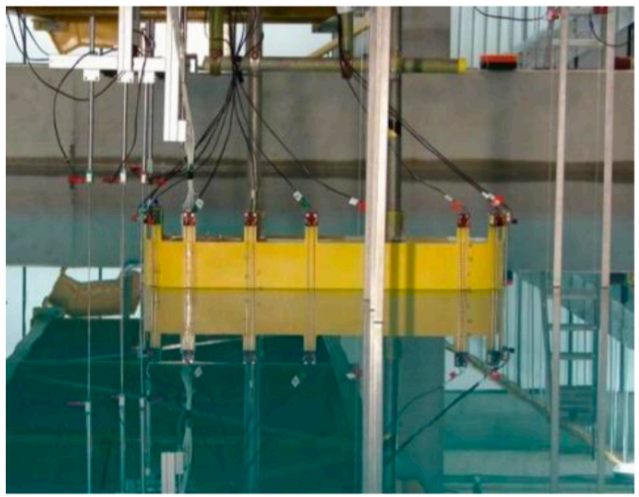
Significant efforts have been made for several decades to develop reliable tools based on fully nonlinear potential flow theory that can handle real geometries of offshore structures and bodies, cf. Karimirad, Michailides & Nematbakhsh (2018). The most mature developments for industrial applications in coastal engineering and offshore engineering are of Boussinesq-type addressing the wave propagation problem, cf. Brocchini (2013), however, these models can conventionally only handle the fluid-structure interaction problem for simple structures and bodies due to the depth-integrated assumption used in the derivation procedures. Even today, handling both the wave propagation problem and the wave-body interaction problem is challenging and is most often considered by hybrid modelling approaches where two different simulation tools are combined through weak coupling.

To resolve these restrictions within a single numerical model, we focus on a complete modelling approach in the setting of potential flow that offer much more flexibility in the numerical discretization procedure. In this work, we propose a SEM-FNPF model, that can handle both the wave propagation problem and the wave-body interaction problem using a multi-element spectral element model that has support for adaptive unstructured meshes with curvilinear elements to represent arbitrarily shaped bodies. The model relies on the recent progress made in two spatial dimensions for a stabilized spectral element method for an Eulerian free surface flow model by Engsig-Karup, Eskilsson & Bigoni (2016) for the wave propagation problem, and Engsig-Karup, Monteserin & Eskilsson (2018) on the stabilization of a Mixed Eulerian-Lagrangian (MEL) formulation. A recent description of state-of-the-art for spectral element methods are given in Ilan, Cantwell, Monteserin, Eskilsson, Engsig-Karup and Sherwin (2018). In this work, we present the new spectral element method in 3D based on an Eulerian formulation. We discard the σ-transformation that was used in Engsig-Karup & Eskilsson (2016). To introduce a surface-piercing fixed body inside the fluid domain, we solve the Laplace problem and discretize this model equation directly and discretize using curvilinear high-order elements based on transfinite interpolation for the free surface layer of elements.

WAVE BASSIN EXPERIMENTS

The experiments we seek to reproduce in this work was conducted as a part of the FROTH project carried out at Plymouth University in the COAST Laboratory facility in UK. In the experiments a fixed scale-model of a FPSO is considered and it is subjected to focused design wave events that are designed to be probable extreme events taken from an irregular sea state based on JONSWAP spectrums. The FPSO has a fixed position with a draft of 0.153 m in still water, a box length of 0.9

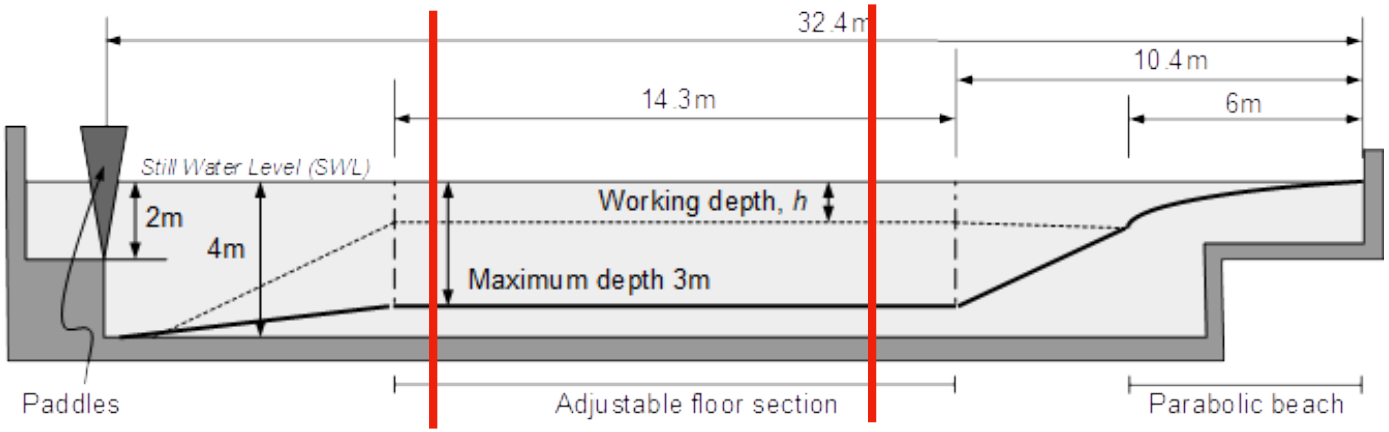




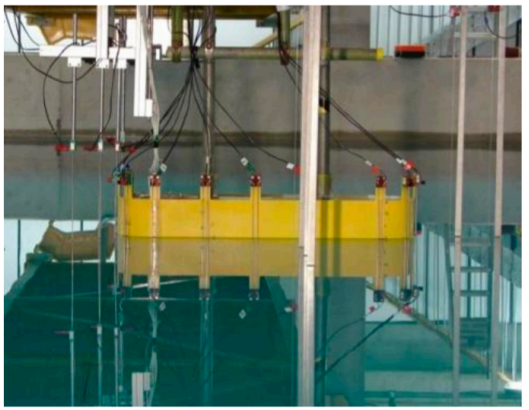
CCP-WSI

A Collaborative Computational Project in Wave Structure Interaction

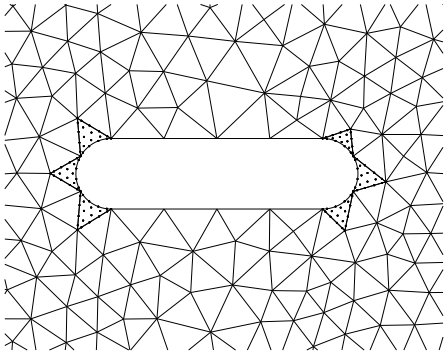
ISOPE 2018 Blindtest Experiment, NWT Setup



Numerical wave tank domain

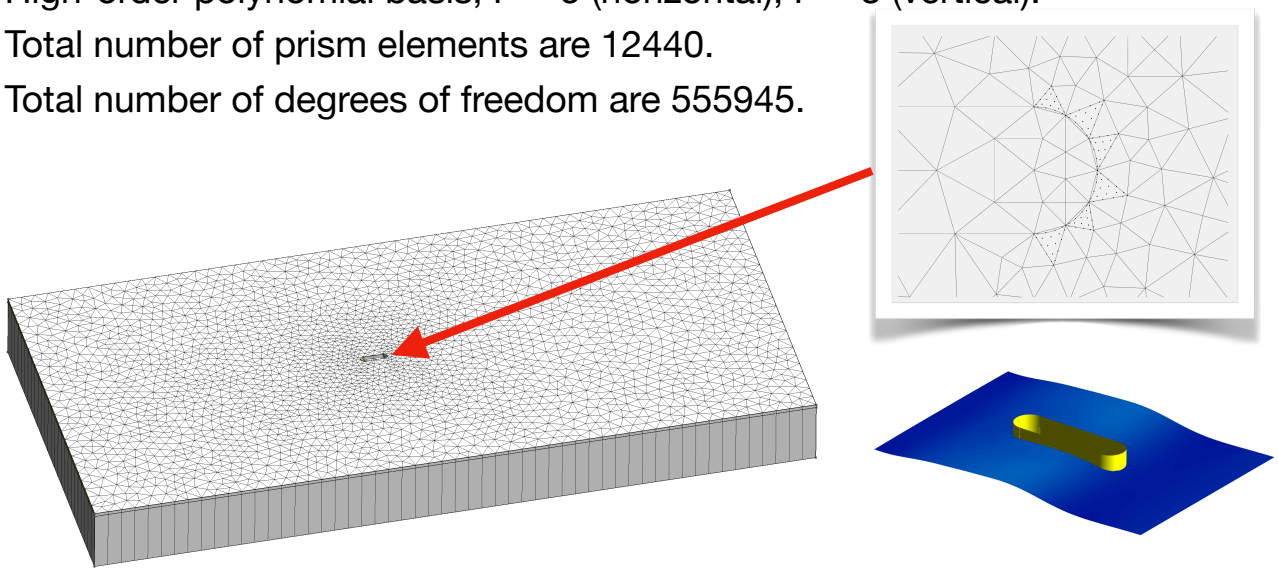


FPSO 2D body structure



FPSO Setup in 3D

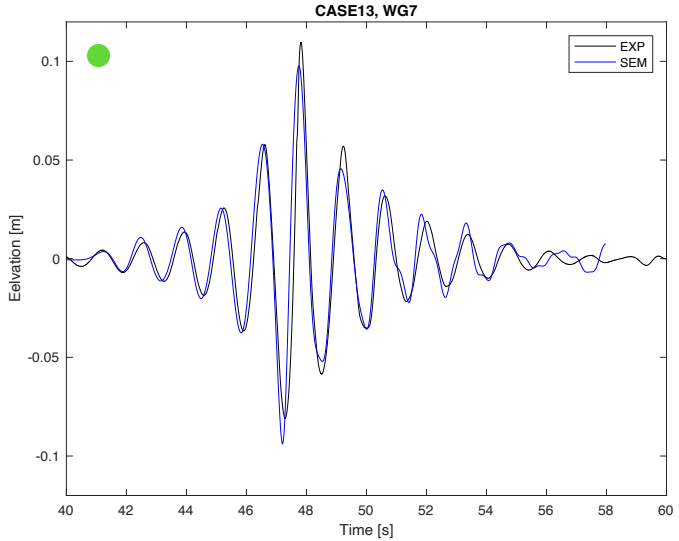
- Domain size:
  - FPSO 2D body structure is resolved with just 14 higher order elements.
  - Length = 12 m, Width = 8 m , h = 2.93 m
- High-order polynomial basis, P = 5 (horizontal), P = 3 (vertical).
- Total number of prism elements are 12440.
- Total number of degrees of freedom are 555945.

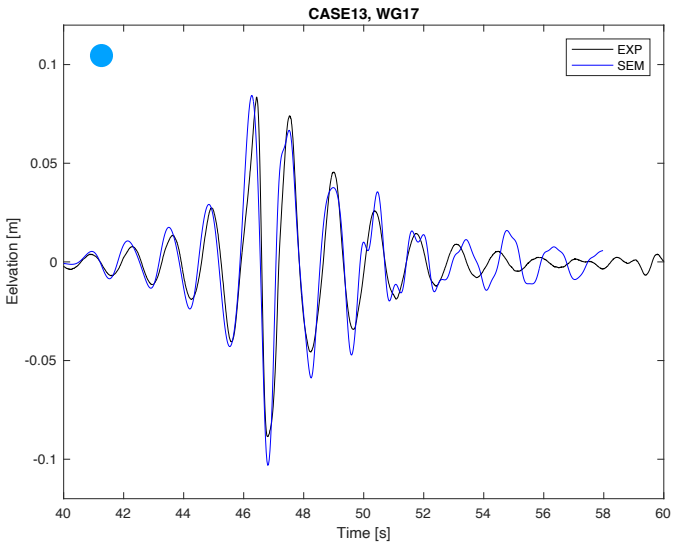


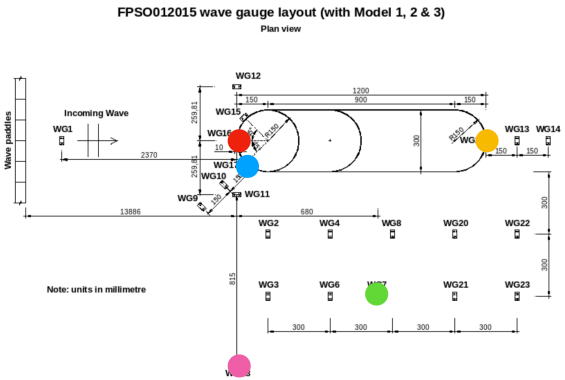
Computational ressources

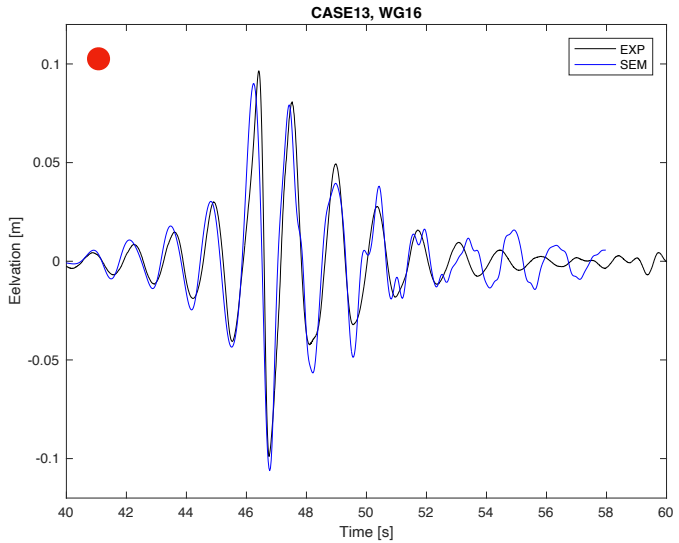
- MarineSEM library v0.8 (beta)
- Matlab R2017a
- Sequential execution.
- Code well structured but not optimised extensively.
- 1 Core, <4.5 days of execution for one simulation.

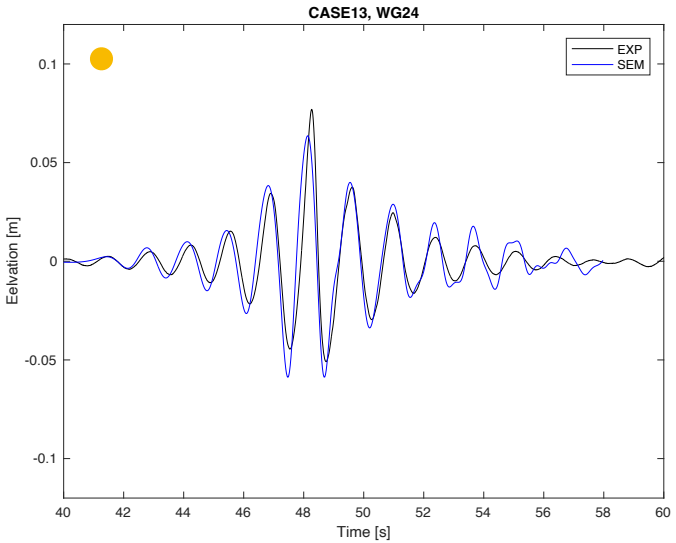
Free Surface Elevation Wave Gauges, Case 1.3 (Fixed FPSO)











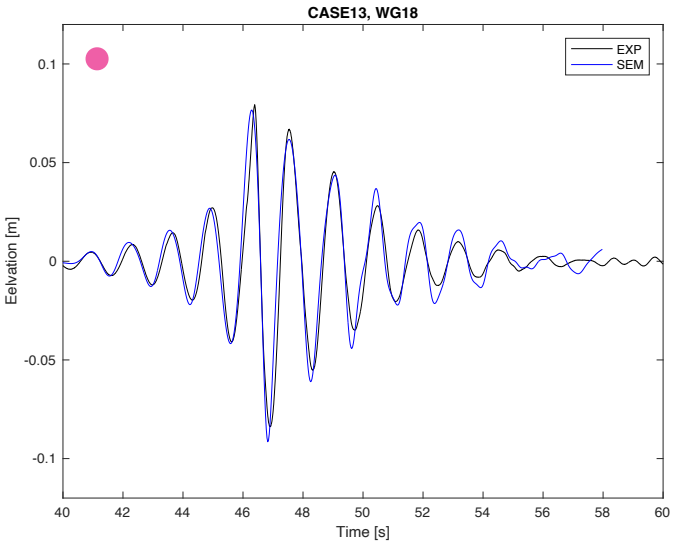
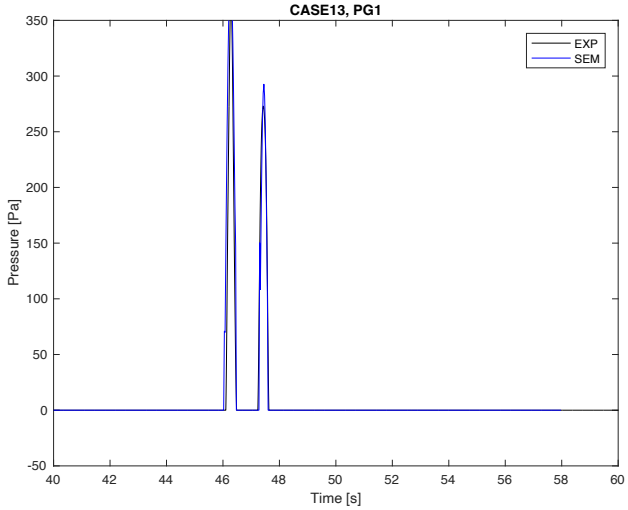
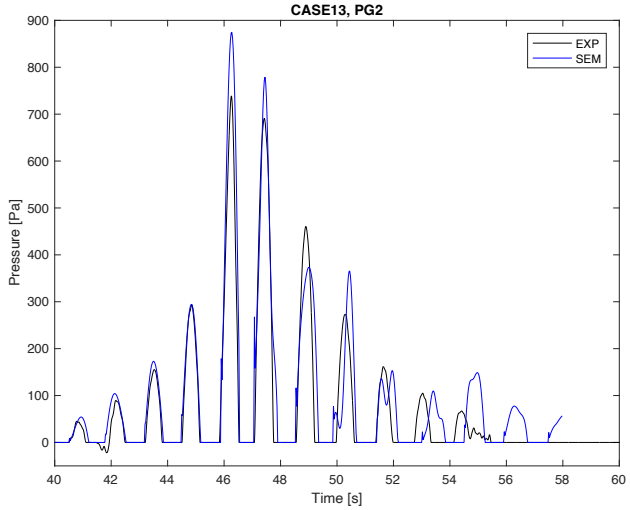
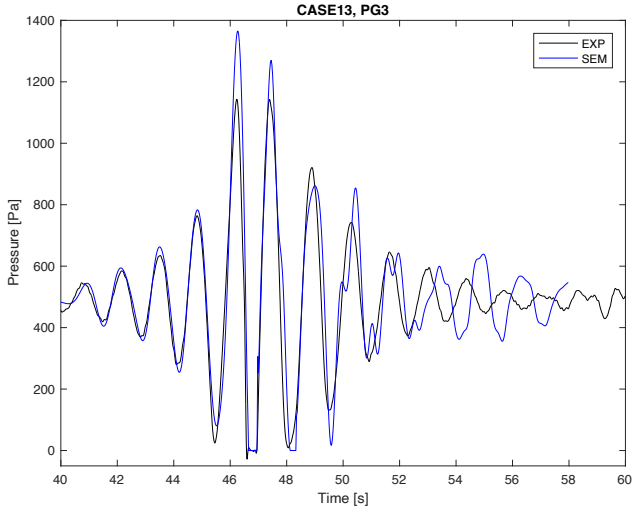


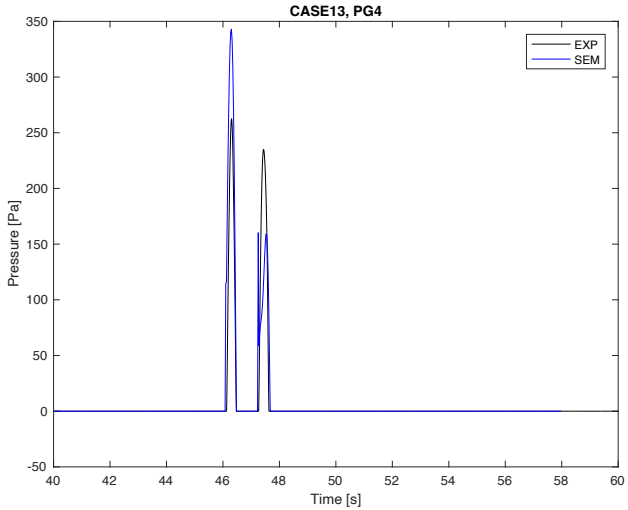
Figure 8. Compute Wave Gauge measurements for NWT with FPSO. The time is started at t=38s of the measurement signal and corresponds to t=0s in the figures.

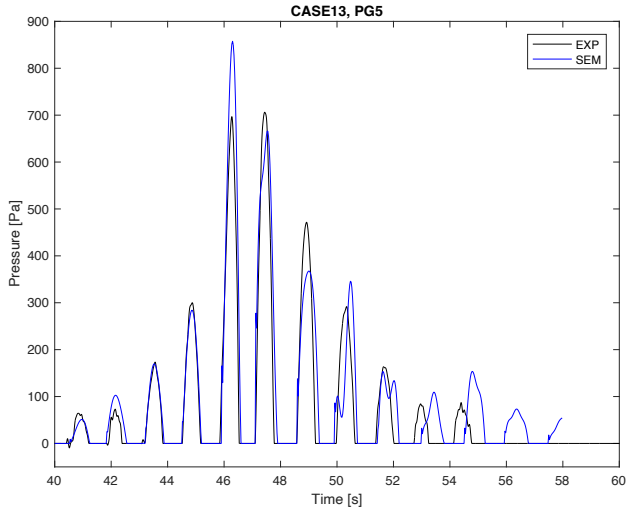
Pressure Gauges, Case 1.3 (Fixed FPSO)

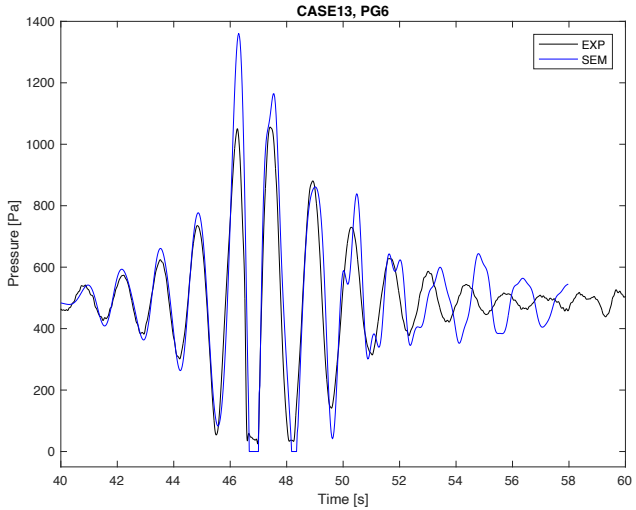












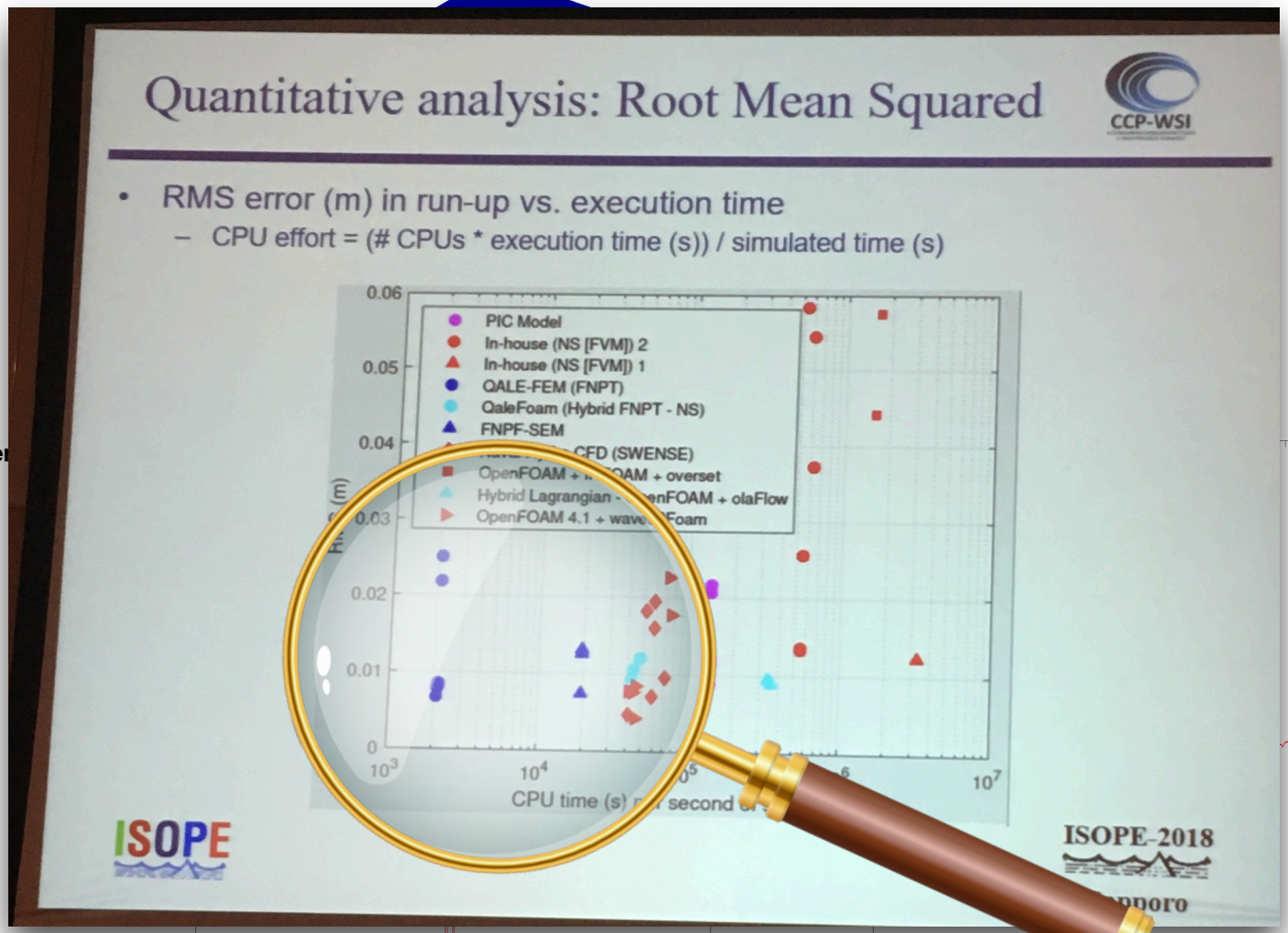
CCP-WSI

A Collaborative Computational Project in Wave Structure Interaction

Simulation results, Wave-body interaction with a fixed FPSO (3D), Case 1.3

time = 38.23

Comparison to referer
WG1 and WG2



New iterative Laplace solver: p-Multigrid Spectral Element Method

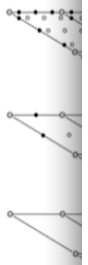
Accelerate convergen **tioned M) :**

$$\mathcal{M}^{-1}\mathcal{L}\phi_h = .$$

Multi-level approach **ith O(n) efficiency:**

Key components:

1. Transfer operators
2. Choice of coarse grids
3. Relaxation strategy



ork	Solver	Method
g (1997) [32]	GMG	LO FDM
lli (2008) [43]	GMRES + FMA	HO BEM
up, Bingham & Lindberg (2009) [21]	GMRES + GMG	HO FDM
up, Madsen & Glimberg (2012) [18]	PDC + GMG	HO FDM
up (2014) [20]	PDC + GMG	HO FDM
up, Laskowski (2020) [17]	p-GMG	HO SEM

TABLE 1

nts in scalable and efficient multigrid or fast multipole iterative solver strategies for /P in FNPF models employing spatial discretization for the fluid volume. Differ- methods of either low-order (LO) or high-order (HO) are considered. Remark, HO ical schemes that has a convergence rate of at least order 3.

Algorithm 5.1 Geometric p-multigrid

1: **function** MULTIGRIDCYCLE($p, \mathbf{u}_{h,}$
2: $\mathbf{u}_{h,p} \leftarrow \mathbf{u}_{h,p} + \mathcal{S}^{-1}(\mathbf{f}_{h,p} - \mathcal{A}_p\mathbf{u}_{h,}$
3: $\mathbf{r}_{h,p} \leftarrow \mathbf{f}_{h,p} - \mathcal{A}_p u_p$
4: $\mathbf{r}_{h,p-1} \leftarrow \mathcal{R}_p^{p-1}\mathbf{r}_{h,p}$
5: **if** $p = 1$ **then**
6: $\tilde{\mathbf{u}}_{h,p-1} \leftarrow (\mathcal{A}_{p-1})^{-1}\mathbf{r}_{h,p-1}$
7: **else if** $p > 1$ **then**
8: $\tilde{\mathbf{u}}_{h,p-1} \leftarrow$ MULTIGRIDCYCLE
9: $\tilde{\mathbf{u}}_{h,p} \leftarrow \mathcal{P}_{p-1}^{p}\tilde{\mathbf{u}}_{h,p-1}$
10: $\mathbf{u}_{h,p} \leftarrow \mathbf{u}_{h,p} + \tilde{\mathbf{u}}_{h,p}$
11: $\mathbf{u}_{h,p} \leftarrow \mathbf{u}_{h,p} + \mathcal{S}^{-1}(\mathbf{f}_{h,p} - \mathcal{A}_p\mathbf{u}_{h,}$
12: **return** $\mathbf{u}_{h,p} \leftarrow \mathbf{u}_{h,p}$

ioned Defect Correction

$\texttt{tol}, \texttt{rtol}, i_{max}, p_{grid}, \nu_1, \nu_2)$
$\|\mathbf{f}_h\| + \texttt{atol}$ **do**

$_{rid}, 0, \mathcal{A}, -\mathbf{r}_h, \nu_1, \nu_2)$

Received: 18 August 2020 | Revised: 30 March 2021 | Accepted: 8 May 2021
DOI: 10.1002/fld.5011

Check for updates

RESEARCH ARTICLE

WILEY

An efficient p-multigrid spectral element model for fully nonlinear water waves and fixed bodies

Allan P. Engsig-Karup[1] | **Wojciech L. Laskowski**[2]

[1]Technical University of Denmark, Lyngby, Denmark
[2]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Correspondence
Allan P. Engsig-Karup, Technical University of Denmark, Lyngby, Denmark.
Email: apek@dtu.dk

Abstract

In marine offshore engineering, cost-efficient simulation of unsteady water waves and their nonlinear interaction with bodies are important to address a broad range of engineering applications at increasing fidelity and scale. We consider a fully nonlinear potential flow (FNPF) model discretized using a Galerkin spectral element method to serve as a basis for handling both wave propagation and wave-body interaction with high computational efficiency within a single modeling approach. We design and propose an efficient $\mathcal{O}(n)$-scalable computational procedure based on geometric p-multigrid for solving the Laplace problem in the numerical scheme. The fluid volume and the geometric features of complex bodies is represented accurately using high-order polynomial basis functions and unstructured meshes with curvilinear prism elements. The new p-multigrid spectral element model can take advantage of the high-order polynomial basis and thereby avoid generating a hierarchy of geometric meshes with changing number of elements as required in geometric h-multigrid approaches. We provide numerical benchmarks for the algorithmic and numerical efficiency of the iterative geometric p-multigrid solver. Results of numerical experiments are presented for wave propagation and for wave-body interaction in an advanced case for focusing design waves interacting with a floating production storage and offloading. Our study shows, that the use of iterative geometric p-multigrid methods for the Laplace problem can significantly improve run-time efficiency of FNPF simulators.

KEYWORDS

fully nonlinear potential flow, geometric p-multigrid, high-order numerical method, Laplace problem, marine offshore hydrodynamics, spectral element method

1 | INTRODUCTION

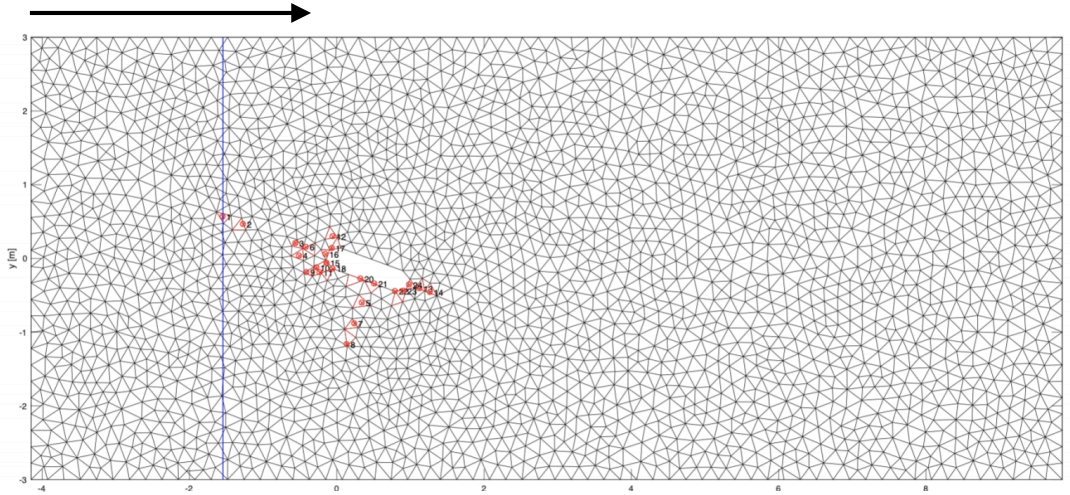
The description of water waves and their nonlinear interaction with bodies are of practical importance in marine offshore hydrodynamics. Few existing phase-resolving simulators are capable of handling both the wave propagation and wave-body interaction problem in a computationally efficient and scalable way, and within the same simulator. Adding such capability is, for example, of relevance in many applications within offshore engineering and increasingly in the renewable sectors. For example, for designing offshore wind turbine foundations,[1] and improving wave energy device simulations.[2] Research in theoretical foundations of wave-body modeling,[3,4] improved uncertainty analysis approaches,[5,6]

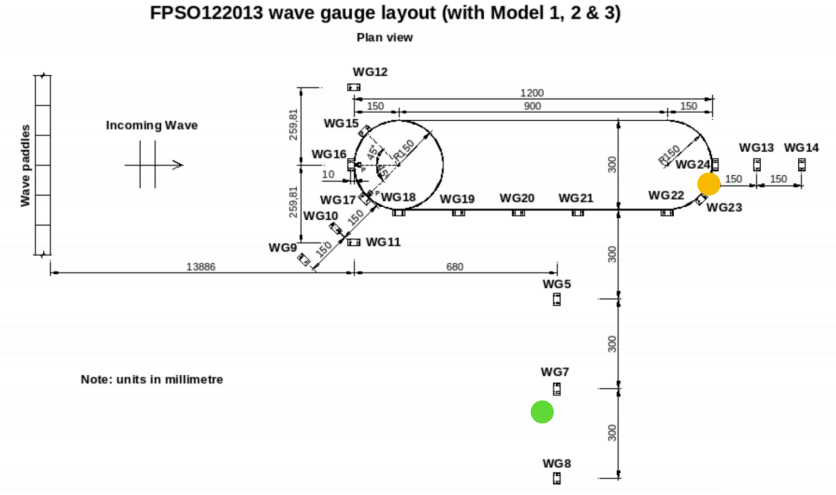
Engsig-Karup, A.P. & Laskowski, W. 2021 Efficient p-Multigrid Spectral Element Model for fully nonlinear water waves and fixed bodies, Int. J. Numer. Meth. Fluids, 93:2823-2841.

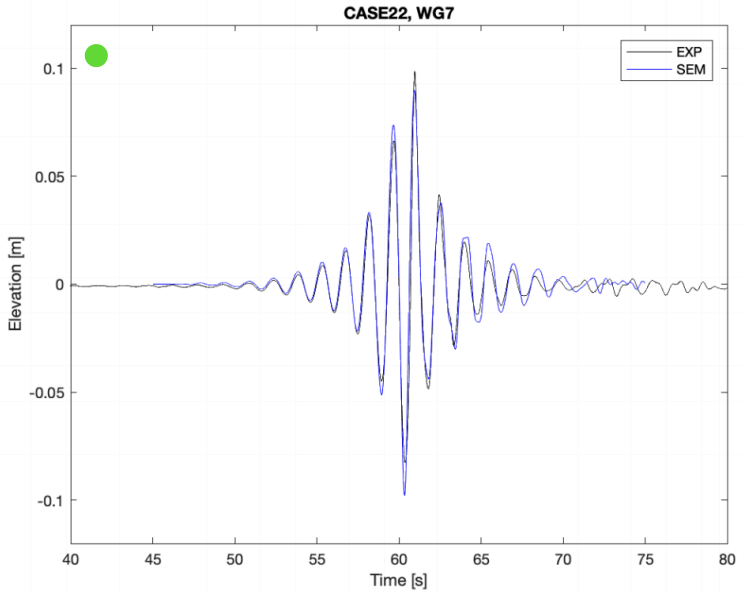
p-Multigrid Spectral Element Method - Case 2.2 results

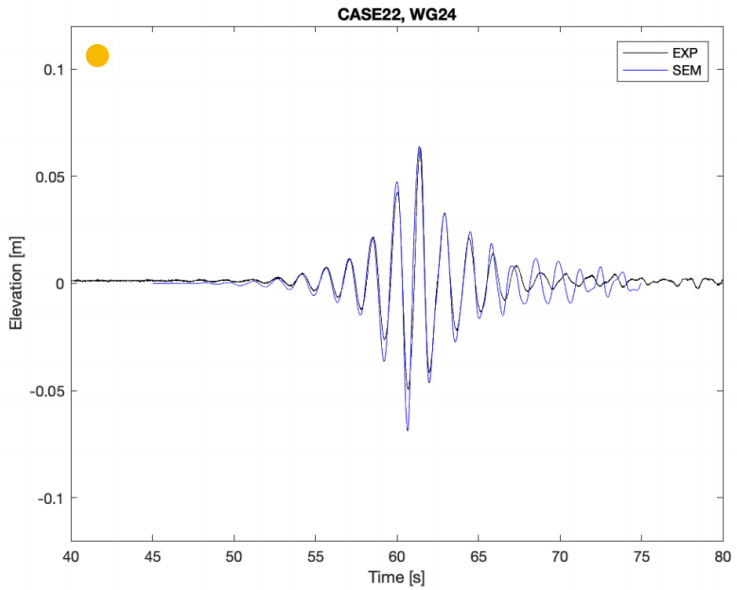
CCP-WSI

A Collaborative Computational Project in Wave Structure Interaction









Engsig-Karup, A.P. & Laskowski, W. 2021 Efficient p-Multigrid Spectral Element Model for fully nonlinear water waves and fixed bodies, Int. J. Numer. Meth. Fluids, 93:2823-2841.

p-Multigrid Spectral Element Method - scaling of work

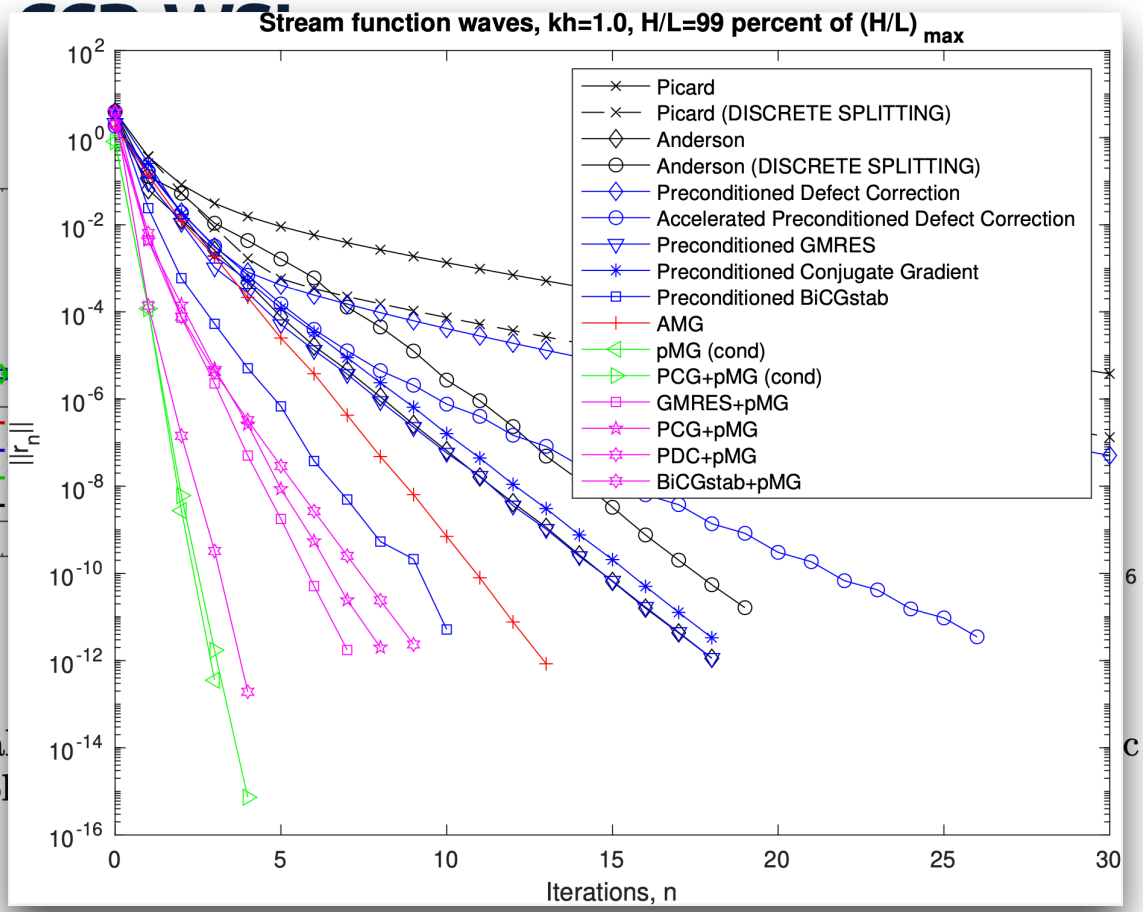
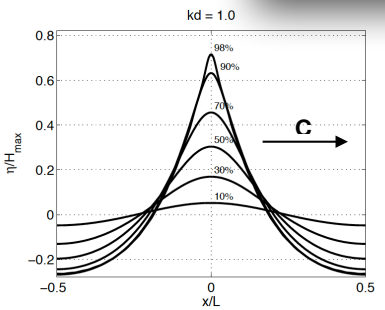


Fig. 9: Scal… c *p*-multigrid sol…

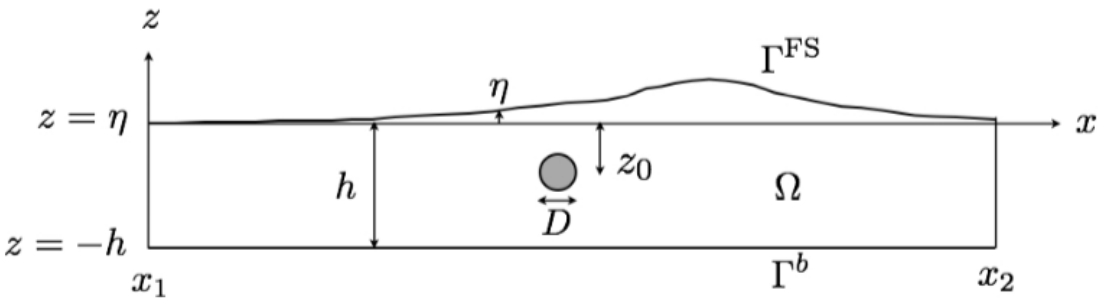


STREAM FUNCTION WAVES

Method	Tolerance 10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
kh = 1	**H/L=10%(H/L)$_{max}$**				**H/L=90%(H/L)$_{max}$**			
p-GMG-cond	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
GMRES + LU	1.80	2.00	2.80	3.00	2.80	3.80	4.80	6.20
PCG + LU	1.80	2.00	2.80	3.00	2.80	3.80	4.84	6.15
PDC + LU	1.80	2.00	2.80	3.60	2.80	4.20	5.80	7.90

Engsig-Karup, A.P. & Laskowski, W. 2021 Efficient p-Multigrid Spectral Element Model for fully nonlinear water waves and fixed bodies, Int. J. Numer. Meth. Fluids, 93:2823-2841.

Mixed-Eulerian Lagrangian formulation



The unsteady nonlinear kinematic and a dynamic free surface boundary conditions expressed in the 'free surface variables only' form as

$$\frac{Dx}{Dt} = \frac{\widetilde{\partial\phi}}{\partial x} = \frac{\partial\tilde{\phi}}{\partial x} - \tilde{w}\frac{\partial\eta}{\partial x}, \quad \frac{D\eta}{Dt} = \tilde{w},$$

$$\frac{D\tilde{\phi}}{Dt} = \frac{1}{2}\left(\left(\frac{\partial\tilde{\phi}}{\partial x}\right)^2 - 2\tilde{w}\frac{\partial\eta}{\partial x}\frac{\partial\tilde{\phi}}{\partial x} + \tilde{w}^2\left(\frac{\partial\eta}{\partial x}\right)^2 + \tilde{w}^2\right) - g\eta,$$

- Discard σ-transform for arbitrary geometries.

The Laplace problem is

$$\nabla^2\Phi = 0, \quad \text{in} \quad \Omega$$

The velocity field inside the fluid volume is determined as

$$(\mathbf{u}, w) = (\nabla, \partial_z)\Phi$$

Water Waves (2019) 1:315–342
https://doi.org/10.1007/s42286-019-00018-5

ORIGINAL ARTICLE

A Mixed Eulerian–Lagrangian Spectral Element Method for Nonlinear Wave Interaction with Fixed Structures

Allan P. Engsig-Karup[1] · Carlos Monteserin[1] · Claes Eskilsson[2,3]

Received: 21 January 2019 / Accepted: 25 September 2019 / Published online: 11 October 2019
© Springer Nature Switzerland AG 2019

Abstract
We present a high-order nodal spectral element method for the two-dimensional simulation of nonlinear water waves. The model is based on the mixed Eulerian–Lagrangian (MEL) method. Wave interaction with fixed truncated structures is handled using unstructured meshes consisting of high-order iso-parametric quadrilateral/triangular elements to represent the body surfaces as well as the free surface elevation. A numerical eigenvalue analysis highlights that using a thin top layer of quadrilateral elements circumvents the general instability problem associated with the use of asymmetric mesh topology. We demonstrate how to obtain a robust MEL scheme for highly nonlinear waves using an efficient combination of (i) global L^2 projection without quadrature errors, (ii) mild modal filtering and (iii) a combination of local and global re-meshing techniques. Numerical experiments for strongly nonlinear waves are presented. The experiments demonstrate that the spectral element model provides excellent accuracy in prediction of nonlinear and dispersive wave propagation. The model is also shown to accurately capture the interaction between solitary waves and fixed submerged and surface-piercing bodies. The wave motion and the wave-induced loads compare well to experimental and computational results from the literature.

Keywords Nonlinear and dispersive free surface waves · Wave–structure interaction · Fully nonlinear potential flow · Spectral element method · Mixed Eulerian–Lagrangian · Marine hydrodynamics · Numerical wave tank

✉ Carlos Monteserin
cmonteserin@hotmail.co.uk
Allan P. Engsig-Karup
apek@dtu.dk
Claes Eskilsson
claes.eskilsson@ri.se

[1] Department of Applied Mathematics and Computer Science, Center for Energy Resources Engineering, Technical University of Denmark, Lyngby, Denmark
[2] Maritime Research, Research Institutes of Sweden, Gothenburg, Sweden
[3] Department of Civil Engineering, Aalborg University, Aalborg, Denmark

Springer

Mixed-Eulerian Lagrangian formulation for wave-body

Proceedings of the 37th International Conference on Ocean, Offshore and Arctic Engineering
OMAE 2018
June 17-22, 2018, Madrid, Spain

77692

NONLINEAR WAVE-BODY INTERACTION USING A MIXED-EULERIAN-LAGRANGIAN SPECTRAL ELEMENT MODEL

Carlos Monteserin
Department of Applied Mathematics and Computer Science
Technical University of Denmark
e-mail: cmonteserin@hotmail.co.uk

Allan P. Engsig-Karup *
Department of Applied Mathematics and Computer Science
Center for Energy Resources Engineering
Technical University of Denmark
Matematiktorvet, Building 303b, Room 108
DK-2800 Kgs. Lyngby (Denmark)
e-mail: apek@dtu.dk

Claes Eskilsson
Research Institutes of Sweden
e-mail: claes.eskilsson@ri.se
Department of Civil Engineering
Aalborg University
e-mail: cge@civil.aau.dk

ABSTRACT

We present recent progress on the development of a new fully nonlinear potential flow (FNPF) model for estimation of nonlinear wave-body interactions based on a stabilised unstructured spectral element method (SEM). We introduce new proof-of-concepts for forced nonlinear wave-body interaction in two spatial dimensions to establish the methodology in the SEM setting utilising dynamically adapted unstructured meshes. The numerical method behind the proposed methodology is described in some detail and numerical experiments on the forced motion of (i) surface piercing and (ii) submerged bodies are presented.

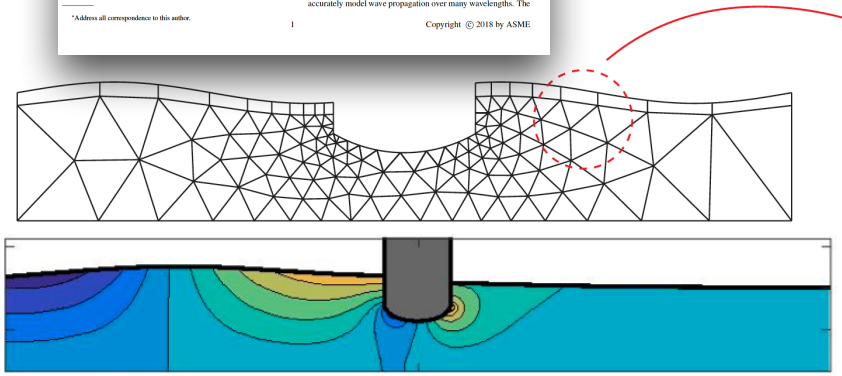
INTRODUCTION

Accurate simulations of nonlinear wave-induced loads on structures, and the subsequent nonlinear wave-body interaction for moving bodies, are of key importance in applications related to ocean engineering and naval architecture. The nonlinear effects are known to be important in cases of more extreme loads and motions, and these are essential to account for in design of bodies or structures. Examples are e.g. design loads for ultimate limit strength (ULS) on marine vessels or other offshore structures, and design of wave energy devices working in resonance.

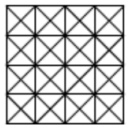
The standard engineering tools for wave-body interaction are based on linear radiation-diffraction theory or strip theory. Lately the use of computational fluid dynamics (CFD) have been put forward for these cases [1] but CFD is very computational demanding for wave propagation. Models based on fully nonlinear potential flow equations (FNPF) are known to be more computationally expensive than the simplified models, not only compared to radiation-diffraction models for wave-body interaction problems, but also compared to nonlinear asymptotic wave models. Such numerical models are in widespread use for coastal engineering applications [2] and are typically used for dispersive and nonlinear wave propagation but needs extension to handle nonlinear wave-body interactions. However, compared to CFD that represents the most expensive computational approach to nonlinear wave-body interactions, the FNPF models are relatively cheap, and can thus be viewed as a compromise between accuracy and computational speed.

The design of efficient and flexible numerical methods that can handle both the nonlinear wave propagation and wave-body interactions is non-trivial. Numerical models based on FNPF equations have traditionally been designed to focus on either wave propagation or wave-body interaction. FNPF models for wave propagation typically use high-order methods (e.g. high-order finite differences [3, 4] or spectral methods [5, 6]) in order to gain efficiency due to the lower degrees of freedom needed to accurately model wave propagation over many wavelengths. The

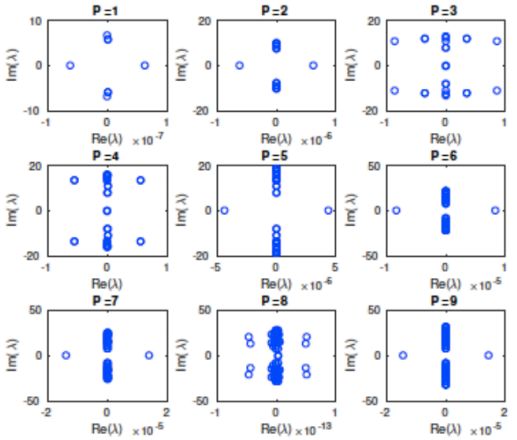
*Address all correspondence to this author.



- MEL formulation for wave-body
- Accelerated Potential formulation needed for body force predictions
- Mesh updates required local and global
- Work-around for the mesh asymmetry problem: use top free surface layer of quads!
- Isoparametric (high-order) representation of the free surface and (if needed) body surface

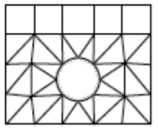


(a) Mesh

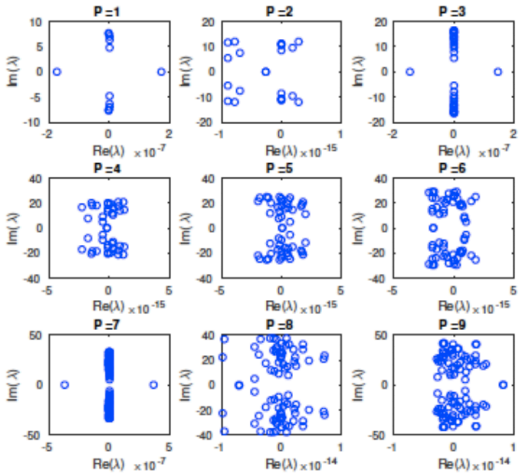


(b) Eigenvalues

Figure 6: Linear stability analysis. Eigenvalues for different polynomial order corresponding to a symmetric structured mesh of triangles. Positive real part of eigenvalues causes temporal instability for polynomial orders 3 and 4.



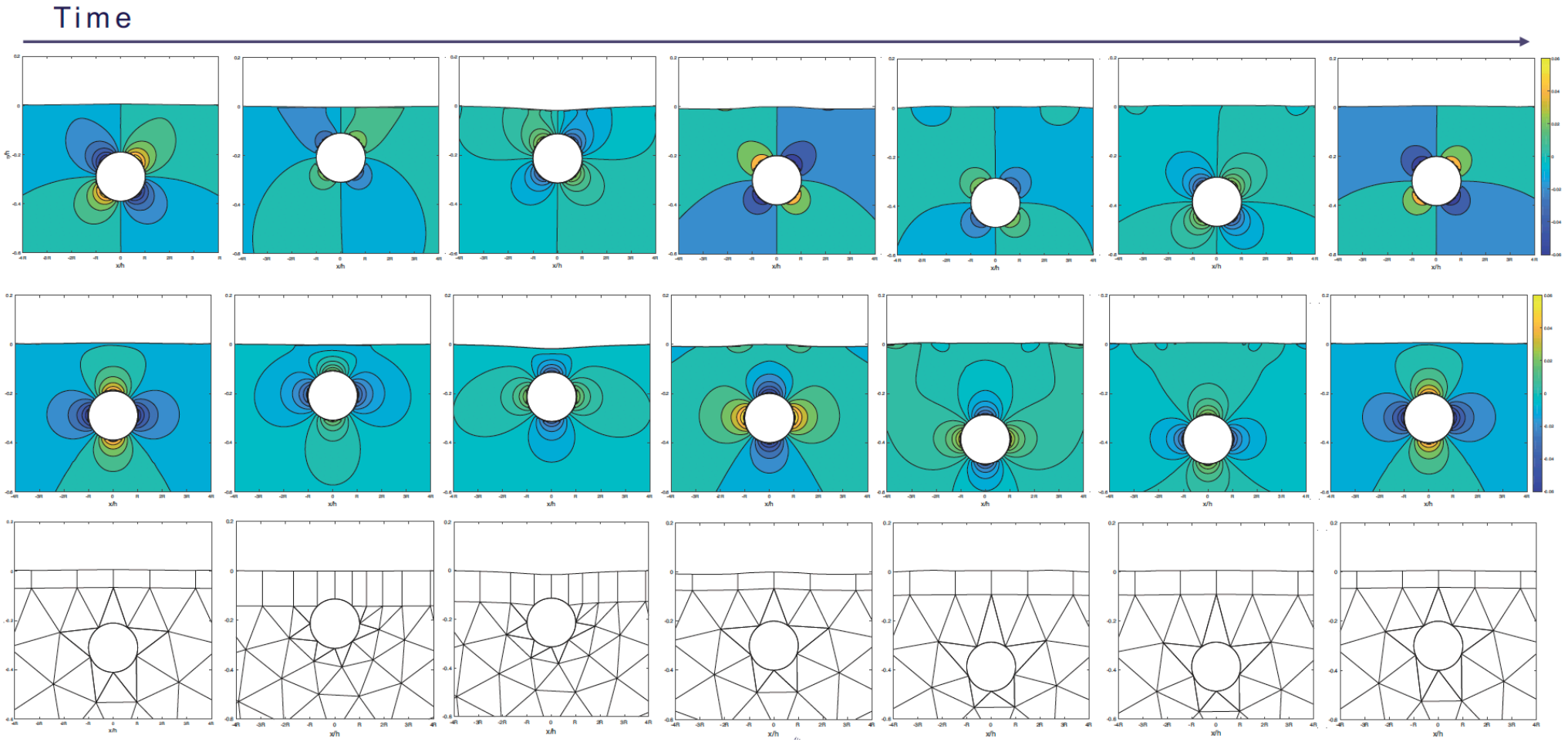
(a) Mesh

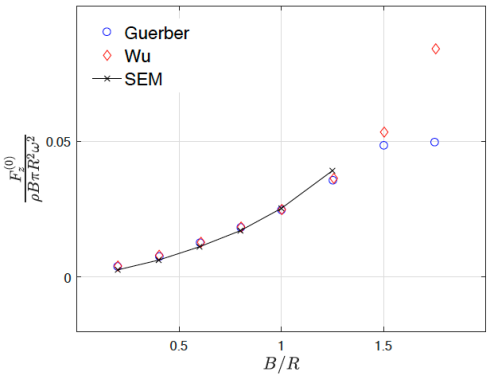


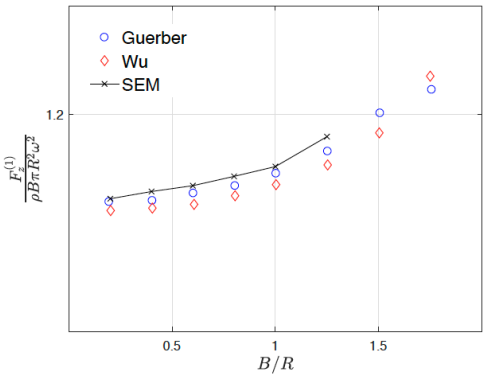
(b) Eigenvalues

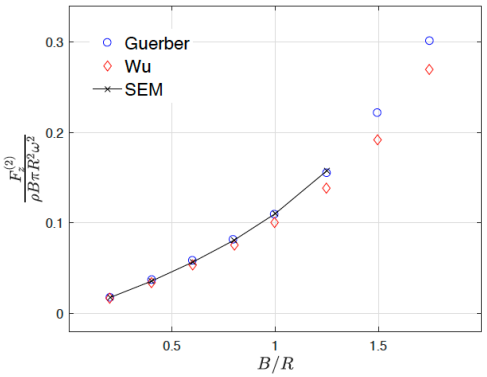
Figure 4: Linear stability analysis. Purely imaginary eigenvalues for different polynomial order to within roundoff accuracy corresponding to hybrid unstructured mesh with submerged circular cylinder.

Heaving submerged cylinder in forced motion (2D)

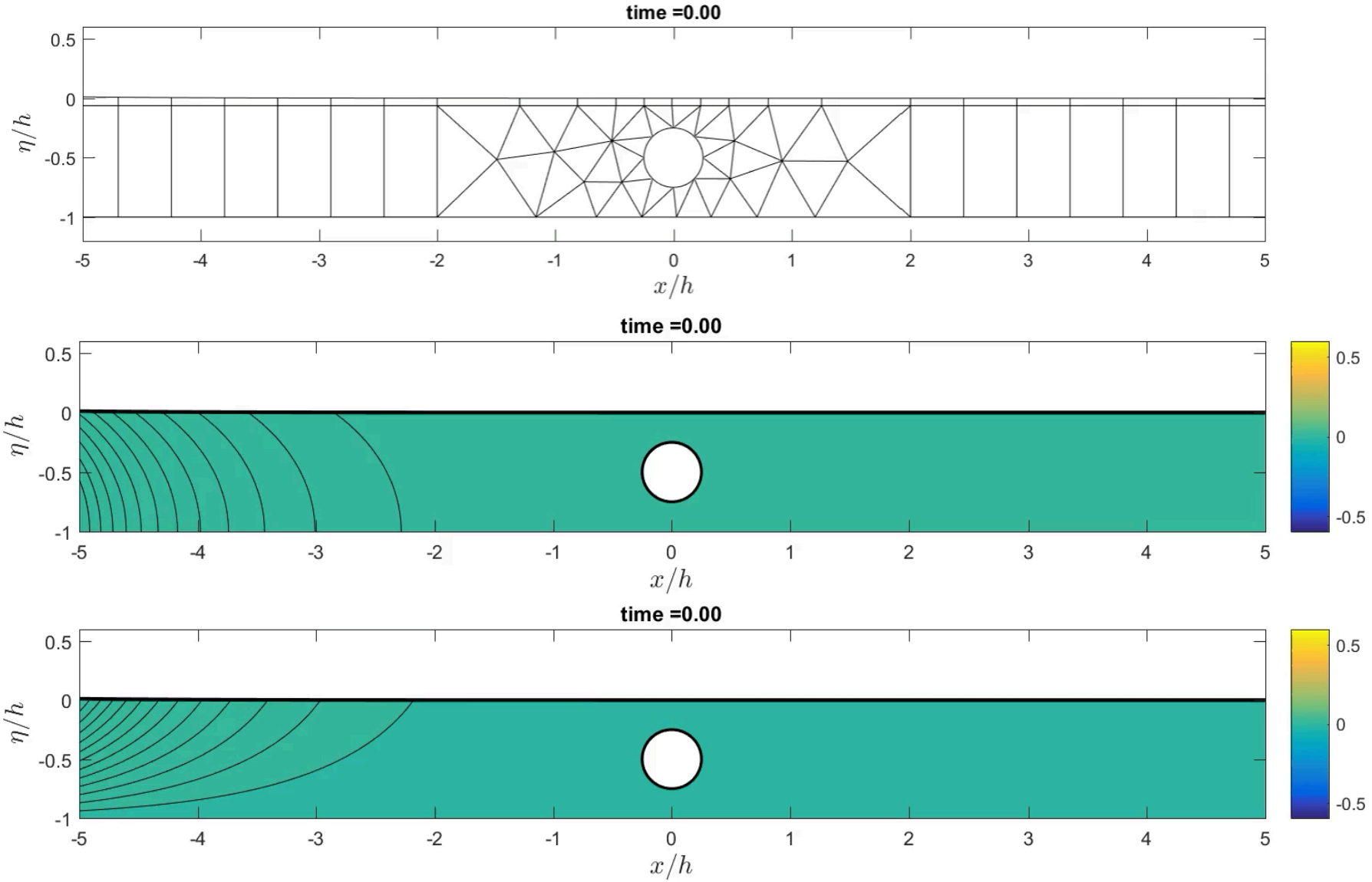




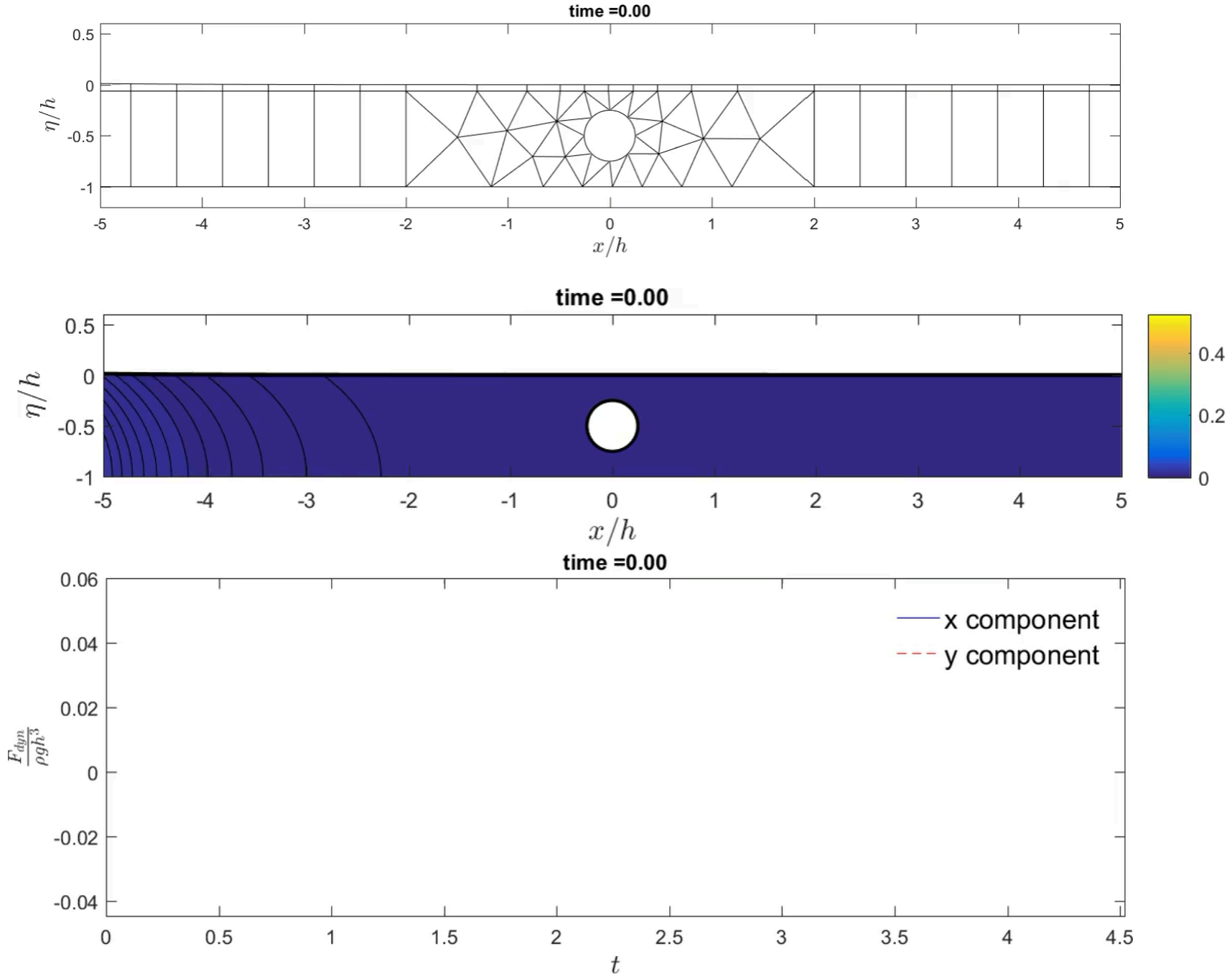




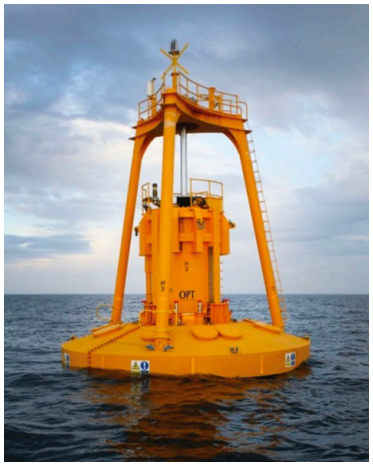
Solitary wave interacting with a submerged cylinder (2D)



Solitary wave interacting with a submerged cylinder (2D)

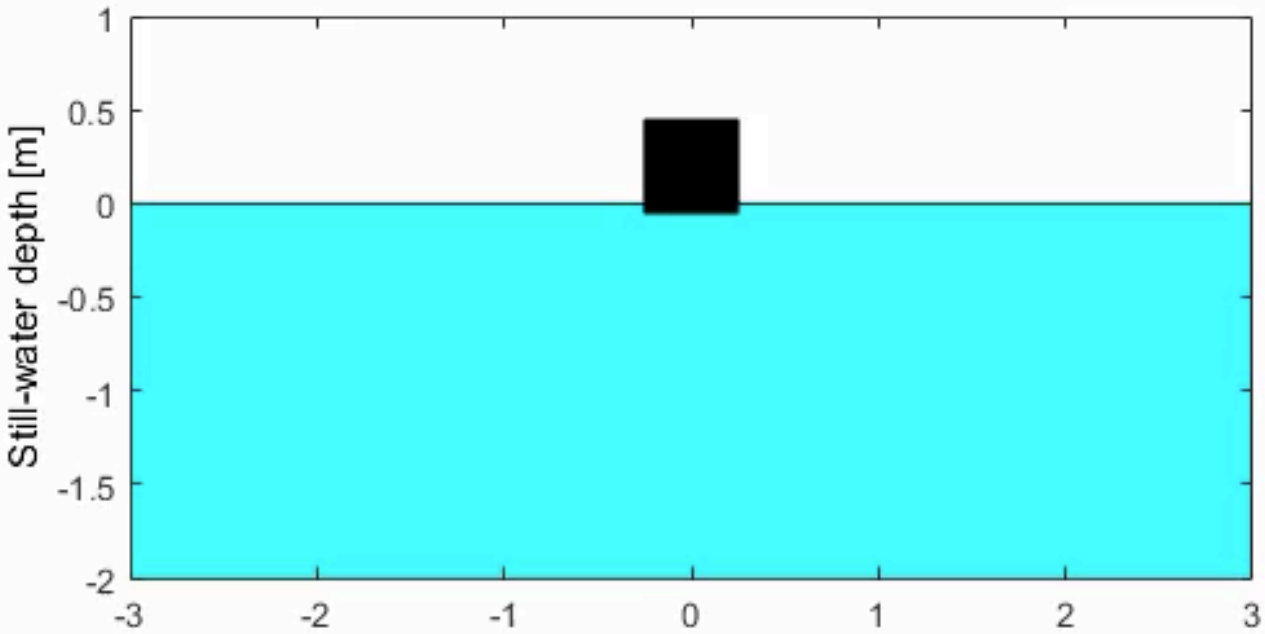


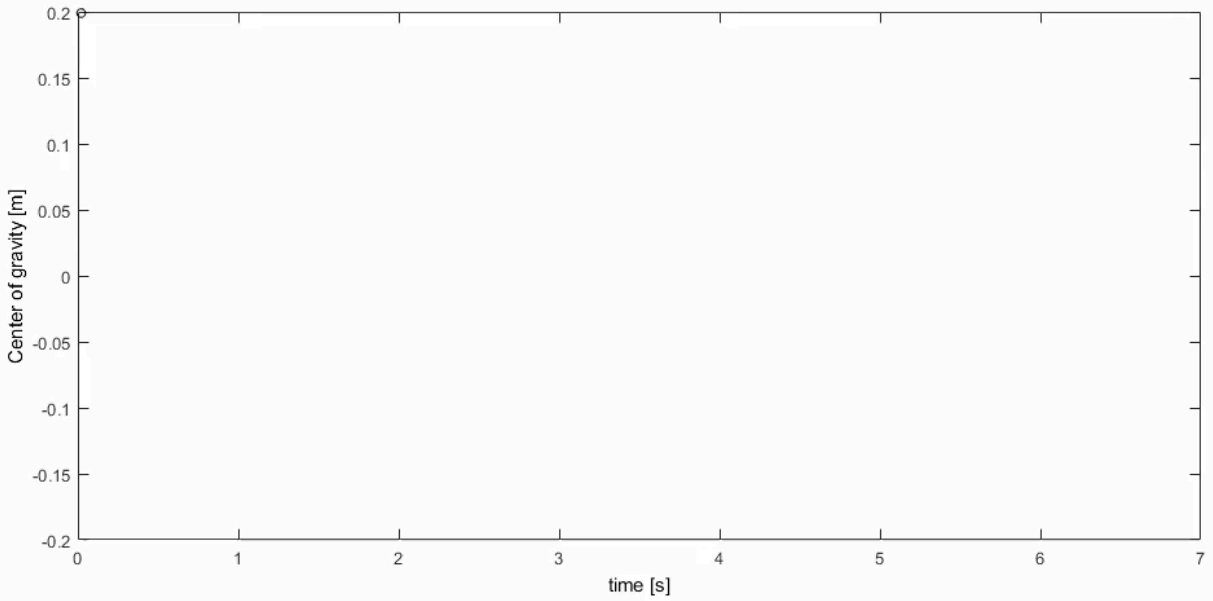
Freely floating body in FNPF-SEM solver - Heave decay test (In progress)

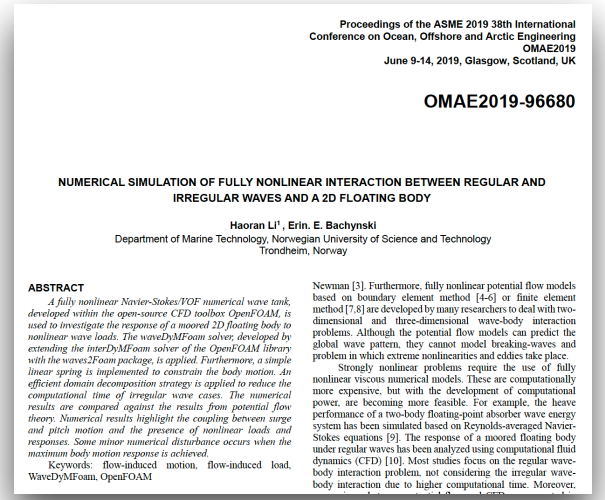


Free floating body movement

- MEL solver
- Coupled body and fluid motion
- Laplace solution gives fluid kinematics
- Acceleration potential method due to Tanizawa (1995) for phi_t calc







Proceedings of the ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering
OMAE2019
June 9-14, 2019, Glasgow, Scotland, UK

OMAE2019-96680

NUMERICAL SIMULATION OF FULLY NONLINEAR INTERACTION BETWEEN REGULAR AND IRREGULAR WAVES AND A 2D FLOATING BODY

Haoran Li[1], Erin. E. Bachynski
Department of Marine Technology, Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT

A fully nonlinear Navier-Stokes/VOF numerical wave tank, developed within the open-source CFD toolbox OpenFOAM, is used to investigate the response of a moored 2D floating body to nonlinear wave loads. The waveDyMFoam solver, developed by extending the interDyMFoam solver of the OpenFOAM library with the waves2Foam package, is applied. Furthermore, a simple linear spring is implemented to constrain the body motion. An efficient domain decomposition strategy is applied to reduce the computational time of irregular wave cases. The numerical results are compared against the results from potential flow theory. Numerical results highlight the coupling between surge and pitch motion and the presence of nonlinear loads and responses. Some minor numerical disturbance occurs when the maximum body motion response is achieved.

Keywords: flow-induced motion, flow-induced load, WaveDyMFoam, OpenFOAM

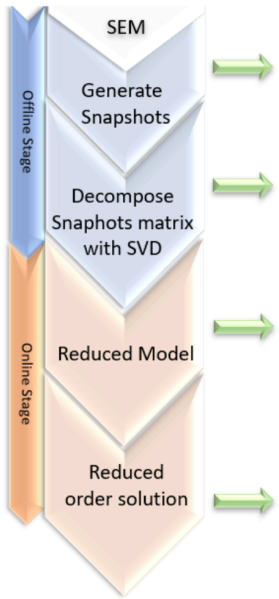
Newman [3]. Furthermore, fully nonlinear potential flow models based on boundary element method [4-6] or finite element method [7,8] are developed by many researchers to deal with two-dimensional and three-dimensional wave-body interaction problems. Although the potential flow models can predict the global wave pattern, they cannot model breaking-waves and problem in which extreme nonlinearities and eddies take place.

Strongly nonlinear problems require the use of fully nonlinear viscous numerical models. These are computationally more expensive, but with the development of computational power, are becoming more feasible. For example, the heave performance of a two-body floating-point absorber wave energy system has been simulated based on Reynolds-averaged Navier-Stokes equations [9]. The response of a moored floating body under regular waves has been analyzed using computational fluid dynamics (CFD) [10]. Most studies focus on the regular wave-body interaction problem, not considering the irregular wave-body interaction due to higher computational time. Moreover,

Reduced Order Modelling for Wave Propagation and Wave-structure interaction

F. G. Eroglu, N. T. Mücke, J. Visbech & A.P. Engsig-Karup

POD-Galerkin for the Laplace Problem



$$\mathcal{S} = [\phi_{j,1}, \phi_{j,2}, ..., \phi_{j,N^S}], \quad \mathcal{S} \in \mathbb{R}^{N^P \times N^S}.$$

We then perform SVD on the matrix, S,

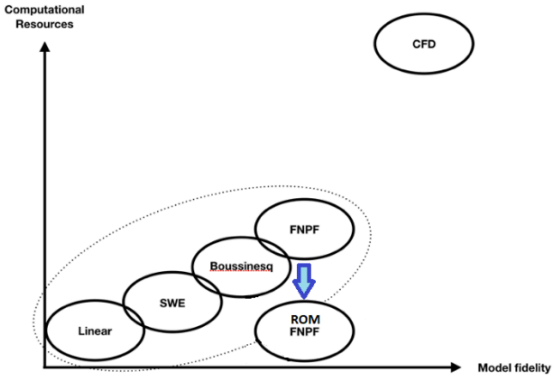
$$\mathcal{S} = \mathcal{U}\Sigma\mathcal{V}^T$$

We use the first l columns of V , and get V_l

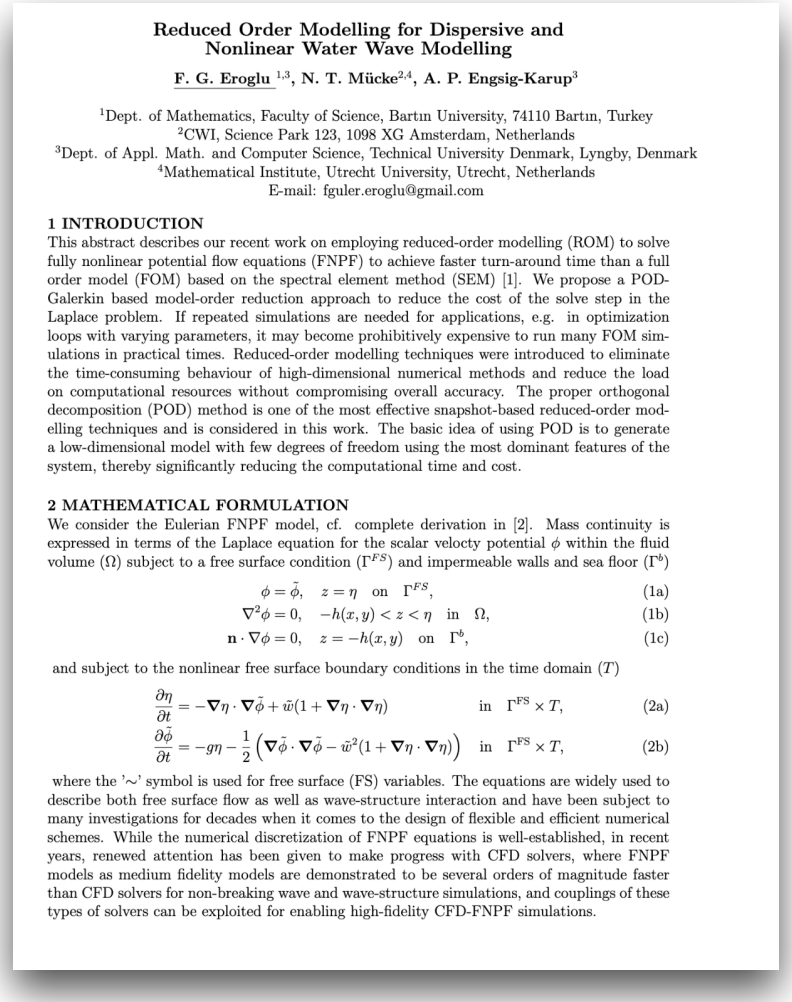
$$\underbrace{\mathcal{V}_l^T \mathcal{L}^* \mathcal{V}_l}_{\text{reduced matrix}} \mathbf{a} = \mathcal{V}_l^T \mathbf{b}^*,$$

$$\phi_i^l(x, z, t) = \mathcal{V}_l \mathbf{a}(t)$$

When ϕ is recovered we continue solving FS equations as usual.



- Accelerated simulation (possibly real-time response)
- Reduces the complexity
- Decreases computational cost
- Contains few DOF



Reduced Order Modelling for Dispersive and Nonlinear Water Wave Modelling

F. G. Eroglu [1,3], N. T. Mücke[2,4], A. P. Engsig-Karup[3]

[1]Dept. of Mathematics, Faculty of Science, Bartın University, 74110 Bartın, Turkey
[2]CWI, Science Park 123, 1098 XG Amsterdam, Netherlands
[3]Dept. of Appl. Math. and Computer Science, Technical University Denmark, Lyngby, Denmark
[4]Mathematical Institute, Utrecht University, Utrecht, Netherlands
E-mail: fguler.eroglu@gmail.com

1 INTRODUCTION

This abstract describes our recent work on employing reduced-order modelling (ROM) to solve fully nonlinear potential flow equations (FNPF) to achieve faster turn-around time than a full order model (FOM) based on the spectral element method (SEM) [1]. We propose a POD-Galerkin based model-order reduction approach to reduce the cost of the solve step in the Laplace problem. If repeated simulations are needed for applications, e.g. in optimization loops with varying parameters, it may become prohibitively expensive to run many FOM simulations in practical times. Reduced-order modelling techniques were introduced to eliminate the time-consuming behaviour of high-dimensional numerical methods and reduce the load on computational resources without compromising overall accuracy. The proper orthogonal decomposition (POD) method is one of the most effective snapshot-based reduced-order modelling techniques and is considered in this work. The basic idea of using POD is to generate a low-dimensional model with few degrees of freedom using the most dominant features of the system, thereby significantly reducing the computational time and cost.

2 MATHEMATICAL FORMULATION

We consider the Eulerian FNPF model, cf. complete derivation in [2]. Mass continuity is expressed in terms of the Laplace equation for the scalar velocty potential ϕ within the fluid volume (Ω) subject to a free surface condition (Γ^{FS}) and impermeable walls and sea floor (Γ^b)

$$\phi = \tilde{\phi}, \quad z = \eta \quad \text{on} \quad \Gamma^{FS}, \tag{1a}$$
$$\nabla^2\phi = 0, \quad -h(x,y) < z < \eta \quad \text{in} \quad \Omega, \tag{1b}$$
$$\mathbf{n} \cdot \nabla\phi = 0, \quad z = -h(x,y) \quad \text{on} \quad \Gamma^b, \tag{1c}$$

and subject to the nonlinear free surface boundary conditions in the time domain (T)

$$\frac{\partial \eta}{\partial t} = -\nabla\eta \cdot \nabla\tilde{\phi} + \tilde{w}(1 + \nabla\eta \cdot \nabla\eta) \quad \text{in} \quad \Gamma^{\text{FS}} \times T, \tag{2a}$$
$$\frac{\partial \tilde{\phi}}{\partial t} = -g\eta - \frac{1}{2}\left(\nabla\tilde{\phi} \cdot \nabla\tilde{\phi} - \tilde{w}^2(1 + \nabla\eta \cdot \nabla\eta)\right) \quad \text{in} \quad \Gamma^{\text{FS}} \times T, \tag{2b}$$

where the '$\sim$' symbol is used for free surface (FS) variables. The equations are widely used to describe both free surface flow as well as wave-structure interaction and have been subject to many investigations for decades when it comes to the design of flexible and efficient numerical schemes. While the numerical discretization of FNPF equations is well-established, in recent years, renewed attention has been given to make progress with CFD solvers, where FNPF models as medium fidelity models are demonstrated to be several orders of magnitude faster than CFD solvers for non-breaking wave and wave-structure simulations, and couplings of these types of solvers can be exploited for enabling high-fidelity CFD-FNPF simulations.

✦ Early work on nonlinear wave propagation, wave-structure and wave-body applications using SEM + ROM.

Reduced Order Modelling for Wave Propagation and Wave-structure interaction

F. G. Eroglu, N. T. Mücke, J. Visbech & A.P. Engsig-Karup

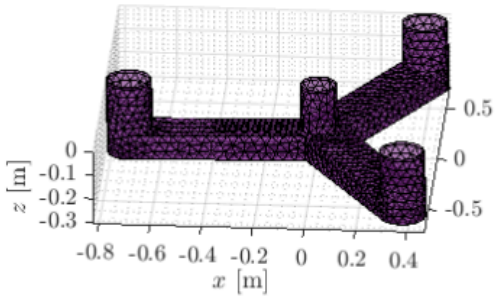
Universiteit Utrecht

CWI
Centrum Wiskunde & Informatica

COWIfonden

CCP-WSI
a Collaborative Computational Project in Wave Structure Interaction

https://www.ccp-wsi.ac.uk/data repository/test cases/ test case 015



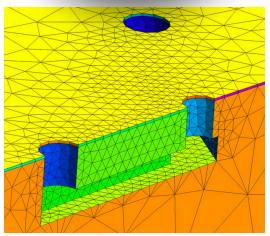


Figure: An example of a two-dimensional surface mesh around Γ^{body} for the floating offshore wind turbine without symmetry boundaries.

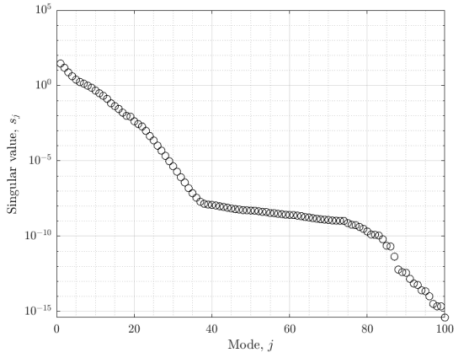


Figure: Singular values

- $DOF = 47,347$ with polynomial order $P = 3$.
- The structure is meshed by $1,269$ 2D surface elements and $7,428$ 3D elements (815 prisms and 6,613 tetrahedral). Hybrid mesh.
- Original computation time is $2.5h$ on one CPU.
- ROM test is for pitch ($k = 5$) motion. This is rotation around the y-axis.

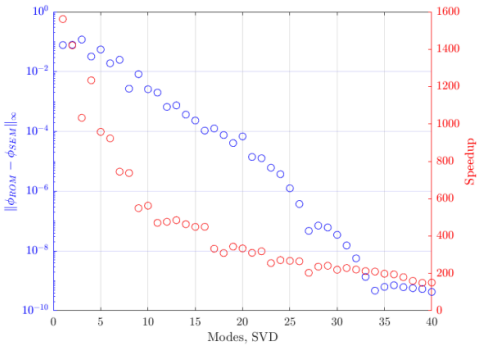
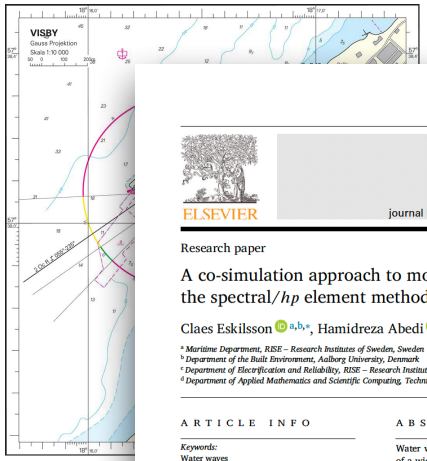


Figure: Convergence tests and Speed estimates

- ✦ Reduced order modeling based on the Spectral Element Method.
- ✦ Ongoing work on nonlinear wave propagation, wave-structure and wave-body applications.
- ✦ In model-scale floating offshore wind turbine test, speed up around 300 with error in machine precision, linear PF.

J. Visbech, H. B. Bingham, C. Eskilsson, J. Palm, and A. P. Engsig-Karup, A High-order Spectral Element based Time-Domain Simulation of a Model-Scale Floating Offshore Wind Turbine, 1st FOWT Comparative Study in connection with ISOPE 2023.

Large scale 3D FNPF simulations



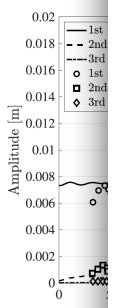


Fig. 11. Whalin semi
monics obtained from
(1971).

Ocean Engineering 340 (2025) 122385

Contents lists available at ScienceDirect

Ocean Engineering

journal homepage: www.elsevier.com/locate/oceaneng

Research paper

A co-simulation approach to modelling fully nonlinear water waves using the spectral/*hp* element method

Claes Eskilsson [a,b,*], Hamidreza Abedi [c], Allan P. Engsig-Karup [d]

[a] Maritime Department, RISE – Research Institutes of Sweden, Sweden
[b] Department of the Built Environment, Aalborg University, Denmark
[c] Department of Electrification and Reliability, RISE – Research Institutes of Sweden, Sweden
[d] Department of Applied Mathematics and Scientific Computing, Technical University of Denmark, Denmark

ARTICLE INFO

Keywords:
Water waves
Fully nonlinear potential flow
Finite element method
High-order
Co-simulation

ABSTRACT

Water wave propagation is of immense importance in the marine environment. We detail the implementation of a widely applicable wave model based on the σ-transformed fully nonlinear potential flow (FNPF) equation implemented within the open-source spectral/*hp* element framework Nektar++. The model is well-suited to describe nonlinear wave propagation in deep waters as well as in nearshore coastal areas where bathymetry effects are important. Spatial discritization is done using the high-order spectral/*hp* element method to keep the numerical dispersion to a minimum, which is important for long-time integration. There are two main novelties in this paper. First, we split the solution of the FNPF equations into two separate models: one to solve the free surface equations in Zakharov form and one to solve the Laplace continuity equation expressed in σ-transformed coordinates. The two solvers are then run as a co-simulation using the CWIPI code coupler, which handles the exchange of free surface variables. This facilitates a highly flexible setup not only in terms of mesh size and polynomial order but also in terms of dividing the available computational resources between the models. Secondly, to avoid having to re-factorize the time-dependent Laplace matrix each time step, we devise a splitting of the Laplace operator into linear and nonlinear parts and solve the Laplace equation in an iterative manner. The present paper outlines the theory and numerical discretization and presents numerical convergence and scaling analyses. In addition, three established benchmark cases are presented, all showing good results.

1. Introduction

Using high performance computing today, it is possible to use fully nonlinear potential flow (FNPF) models for tasks that, due to computational effort, previously fell into the use of phase-averaging wave models. Some examples are the screening of extreme waves (Ducrozet et al., 2007; Wang et al., 2021) and waves propagating in deep fjords (Wang et al., 2022). With increased computational power, FNPF models can also be used instead of depth-integrated coastal wave models, such as models based on Boussinesq-type equations. As shown in Engsig-Karup et al. (2016), the dispersion properties of an FNPF model can be tuned by the spatial resolution in the vertical direction, yielding a numerical equivalence to the analytical expansion used to define the dispersion characteristics of the Boussinesq equations. Thus, FNPF modeling can also become competitive for coastal studies.

Looking at the plethora of numerical models for solving the FNPF equations, e.g. Longuet-Higgins and Cokelet (1978), Dommermuth and Yue (1987), Grilli et al. (2001), Ma and Yan (2006), Engsig-Karup et al. (2009), Wang et al. (2022) and Tong et al. (2024), there is a clear demarcation between models for wave propagation and models for wave-body interaction. Models that target wave propagation and transformation, such as Dommermuth and Yue (1987), Ma and Yan (2006), Engsig-Karup et al. (2009) and Wang et al. (2022), are mainly of high-order. It is not surprising that high-order models with little numerical dispersion are preferable when modeling dispersive waves. An example of a widely used high-order model is the 'high-order spectral' (HOS) model, Dommermuth and Yue (1987) and Ducrozet et al. (2016). HOS is based on expanding the velocity potential in terms of wave steepness to order M, and then solves the wave modes of the free surface conditions by a pseudospectral method. The theory behind HOS was introduced in 1987, but it remains *the* state-of-the-art FNPF model. The today very popular

* Corresponding author.
E-mail addresses: claes.eskilsson@ri.se, claese@build.aau.dk (C. Eskilsson), hamidreza.abedi@ri.se (H. Abedi), apek@dtu.dk (A.P. Engsig-Karup).

https://doi.org/10.1016/j.oceaneng.2025.122385
Received 26 March 2025; Received in revised form 16 July 2025; Accepted 2 August 2025
Available online 10 August 2025
0029-8018/

- Sigma-transformed FNPF
- Nektar++ is an open-source spectral/hp element framework
- Parallel version using MPI
- Iso-parametric mapping, Prism/Hexa hydbrid mesh

Eskilsson, C., Abedi, H., & Engsig-Karup, A. P. (2025). A co-simulation approach to modelling fully nonlinear water waves using the spectral/hp element method. Ocean Engineering, 340,122385.

Cantwell, C. D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., de Grazia, D., Yakovlev, S., Lombard, J.-E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R. M., & Sherwin, S. J. (2015). Nektar++: An open-source spectral/hp element framework. Computer Physics Communications, 192, 205-219.