FACULTY OF ENGINEERING
AND ARCHITECTURE

DEPARTMENT OF INFORMATION TECHNOLOGY
ELECTROMAGNETICS RESEARCH GROUP

BEAST: Boundary ELement Analysis and Simulation Toolkit (and Makeitso)

Kristof Cools – UGent

GHENT UNIVERSITY

This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001847).

Funded by the European Union

Challenges in Scientific Computing

- Storage and loading of simulation data
- Data validity during development
- Long-running scripts can be interrupted:
  - Because of bugs
  - Because of lack of computational resources
  - Because of power outages
  - …
- Parameter sweeps:
  - Do not recompute iterations that have been done before
  - Store the parameters together with the results
- Share results common to different simulations

Challenges in Boundary Element Methods

- Reformulation of BVP as Integral equation. E.g scattering by perfect conductor:

$$Tu = -n \times e^{inc}$$

$$(Tu)(x) = -i\kappa n \times \int_{\Gamma} \frac{e^{-i\kappa|x-y|}}{4\pi|x-y|} u(y)dy + \frac{1}{i\kappa} n \times grad \int_{\Gamma} \frac{e^{-i\kappa|x-y|}}{4\pi|x-y|} div_{\Gamma} u(y)dy$$

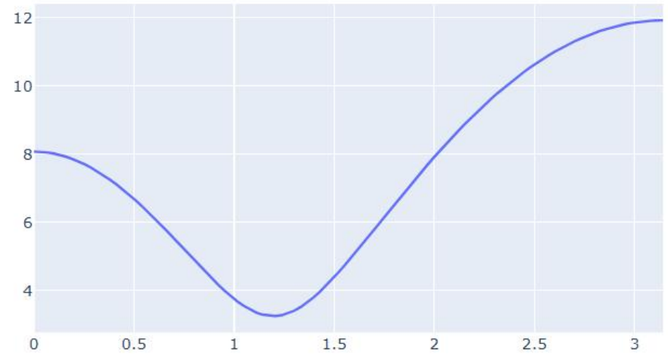
- Main difference with finite element methods:
  - Dense matrices:
    - compression needed (H-matrix, FMM)
    - iterative solution
    - preconditioner required
  - Complicated integrands require fine-grained control over quadrature methods

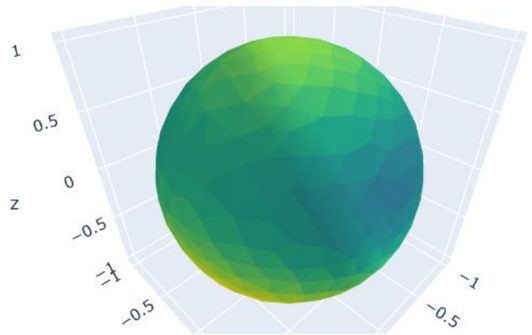
Challenges in domain decompositions

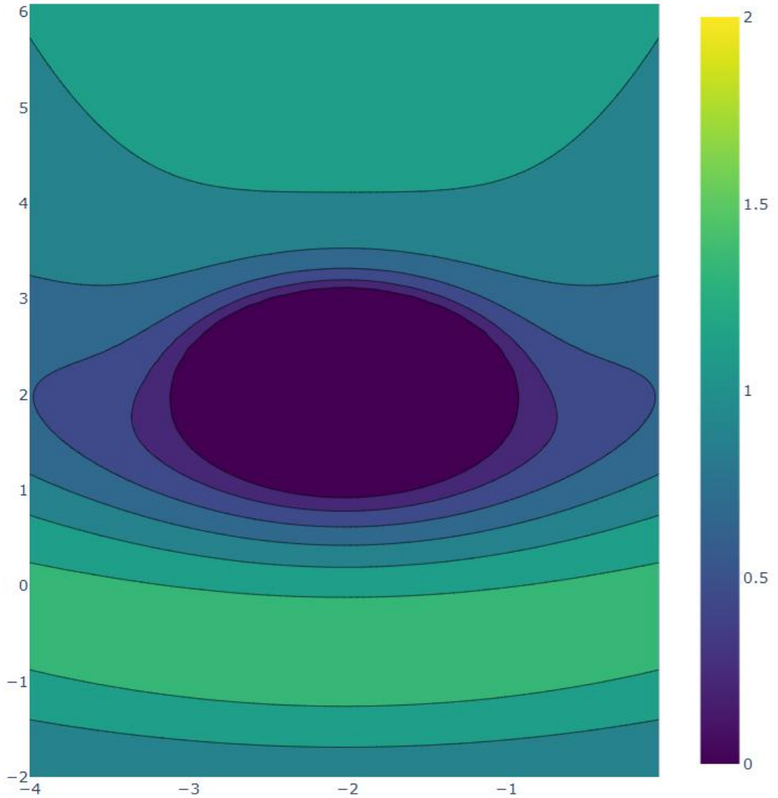
- Complicated multi-level block structure of the global system
- Storage can vary from one block to another:
  - Dense vs sparse
  - Space-time vs space only
  - Compressed vs uncompressed
- Many block positions are unused and should not be filled by zeros
- Pairs of Cauchy data per domain
- Mixed formulations introduce Lagrange multipliers
- Mix of algebraic preconditioners and Calderon preconditioners
- Unified syntax for the solution of linear systems
- Separation of formulation, discretization, and assembly

BEAST: Boundary Element Analysis and Simulation TK

```julia
using CompScienceMeshes
using BEAST

# --- basis functions
Γ  = meshsphere(1.0, 2.5)   # triangulate sphere of radius one
RT = raviartthomas(Γ)       # define basis functions

# --- operators & excitation
T = Maxwell3D.singlelayer(wavenumber=2.0)                              # integral operator
E = Maxwell3D.planewave(direction=x̂, polarization=ẑ, wavenumber=2.0)  # excitation
e = (n × E) × n # tangential part

# --- compute the RHS and system matrix
e = assemble(e, RT)         # assemble RHS
T = assemble(T, RT, RT)     # assemble system matrix

# --- solve
u = T \ -e
```







GHENT UNIVERSITY

BEAST: Main features

- All operators that appear in the integral equation based analysis of:
  - Helmholtz 2D == Maxwell 3D
  - Helmholtz 3D
  - Maxwell 3D
  - Laplacian 2D/3D
- The corresponding potential operators for field reconstruction
- Full control over quadrature methods
- Lowest order and higher order scalar and vectorial basis functions
- Dual basis functions for building preconditioners and second kind methods
- Support for the space-time Galerkin discretization of time-domain integral equations
- Integration with IterativeSolver.jl using wrappers implementing LinearMaps.jl
- Domain decomposition language facilitating domain decomposition and coupling to FEM

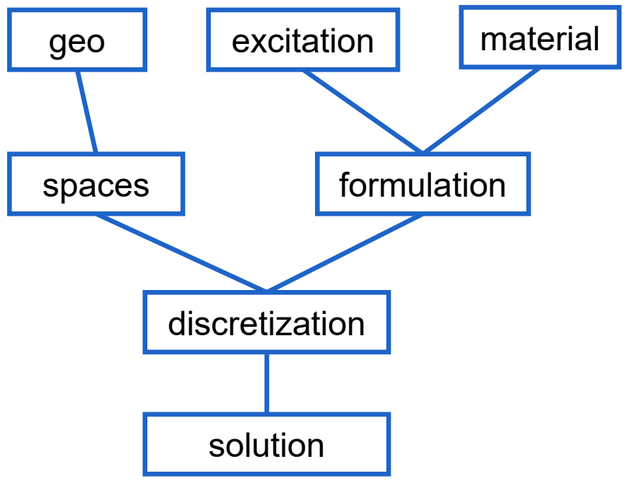
Declarative Programming

- Definition: the programmer describes what should be computed. The framework figures out how to compute this with minimal resources.
- Inspiration: GNU make: description of complicated software projects as a dependency graph of sources files, object files, libraries, and executables
- Differences:
  - Time stamp validity checks too finicky in a multi-threaded universe
  - Recipes to build targets (aka products) from dependencies can be much more complicated and can change in time.
  - Targets depend on other targets but also on parameters. Sweeps are common in scientific computing.

Makeitso.jl

- A simulation is a tree of targets
- Targets are a combination of recipes (=code) on how to compute them from other targets and can depend on parameters
- When a target is requested (make(target; pars…)):
  - A valid target is looked for in memory: if it is there, no work required!
  - A valid target is looked for on disk: if it is there, it is loaded.
  - Else, the target recursively requests its dependencies to be produced, then computes the result following the recipe
- Validity: a hash is computed from the target recipe and the hashes for all its dependencies. This hash has to coincide with that stores with existing results.
- Storage location: file path composed of:
  - The data directory
  - The relative path of the file that defines the target in the project
  - The name of the target decorated by its hash
  - The filename contains the parameters and their hash
- File format: .jld2, a dialect of hdf5 (just like Matlab .mat files)
- Simulations are dependency trees that can be composed by including targets defined in various files: mix-and-match of geometries, excitations, materials, solutions methods, post-processing routings, and visualization is possible
- Sweeps are targets for the recipe is repeated over the Cartesian product of one or more parameter ranges. It is possible to define dependencies that are independent of the swept over parameters and that should be computed only once.
- Sometimes (e.g. compare a set of solutions against a single reference solution), a target at a specific value of the parameter plays a special sole. This can be supported by allowing for a parameter transformation step to be included before the recipe is called.

Example: Convergence of the EFIE



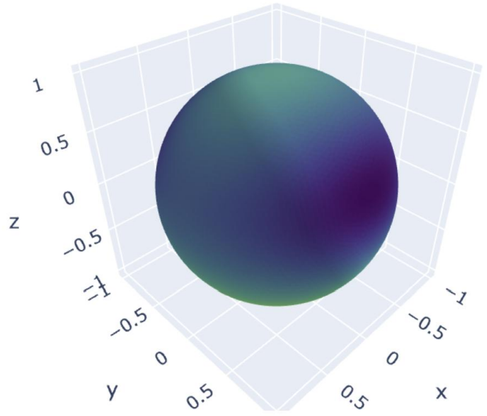

```julia
@target geo (;h) -> begin

    G10 = meshsphere(radius=1.0, h=h)

    return (;∂Ω=[-G10, G10])
end
```

```julia
@target excitation (material,; κ) -> begin

    Einc = Maxwell3D.planewave(direction=ẑ/√2, polarization=ŷ, wavenumber=material[1].κ)
    Hinc = -1/(im*material[1].κ*material[1].η)*curl(Einc)

    exc = Vector{Any}(undef, length(material))
    fill!(exc, (;Einc=nothing, Hinc=nothing))
    exc[1] = (;Einc=Einc, Hinc=Hinc)

    return exc
end
```

```julia
@target material (numdoms, ;κ) -> begin
    return fill((;κ=κ, η=1.0), numdoms)
end
```

```julia
@target spaces (geo) -> begin
    X = raviartthomas(geo.∂Ω[1])
    (;X)
end
```

```julia
@target formulation (material, excitation) -> begin

    @hilbertspace j
    @hilbertspace k

    κ, η = material[1]
    T = Maxwell3D.singlelayer(wavenumber=κ)

    (;Einc) = excitation[1]
    e = (n × Einc) × n

    a = η * T[k,j]
    l = e[k]

    return (;bilforms=(;a), linforms=(;l))
end
```




```julia
@target memonly=true discretization (formulation, spaces) -> begin
    a = formulation.bilforms.a
    l = formulation.linforms.l
    X = spaces.X

    A = BEAST.assemble(a, X, X)
    b = BEAST.assemble(l, X)
    (;matrices=(;A), vectors=(;b))
end
```

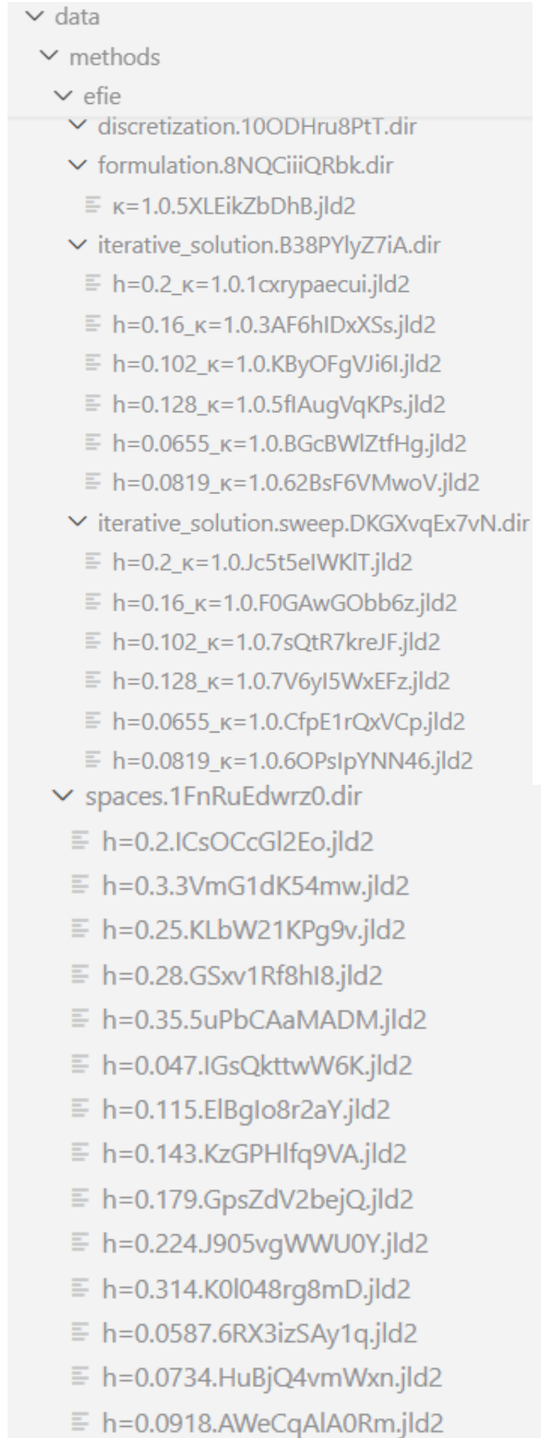
```julia
@target solution (discretization, spaces) -> begin
    A = discretization.matrices.A
    b = discretization.vectors.b

    A = BEAST.BlockArrays.BlockedArray(Matrix(A), axes(A))

    A⁻¹ = BEAST.lu(A)
    u = A⁻¹ * b

    (;u=BEAST.FEMFunction(u, spaces.X))
end
```

GHENT UNIVERSITY

Example: convergence of the EFIE

- Outputs are stored:
  - In a dir based on target definition site and hash of the full recipe
  - In a file based on the parameter values and their hash
- I different simulations share a subtree of the dependency tree:
  - results are automatically shared
- Best practice:
  - Define each recipe in its own module:

```julia
using Makeitso

using PlotlyJS
using BEAST

module Sim1
include("../problems/p1_geo1_mat.jl")
include("../methods/MxMFIE.jl")
include("../postprocessing/nearfield.jl")
include("../figures/nearfield.jl")
end

module Sim2
include("../problems/p1_geo1_mat.jl")
include("../methods/EFIE.jl")
include("../figures/nearfield.jl")
end

module Sim3
include("../problems/p1_geo1_mat.jl")
include("../methods/MxEFIE.jl")
include("../postprocessing/nearfield.jl")
include("../figures/nearfield.jl")
end

cfg = (;h=0.03, κ=2π/3.0, η=1.0)
# make(Sim1.formulation; cfg...)
u1 = make(Sim1.solution; cfg...).u
u2 = make(Sim2.solution; cfg...).u
u3 = make(Sim3.solution; cfg...).u

```



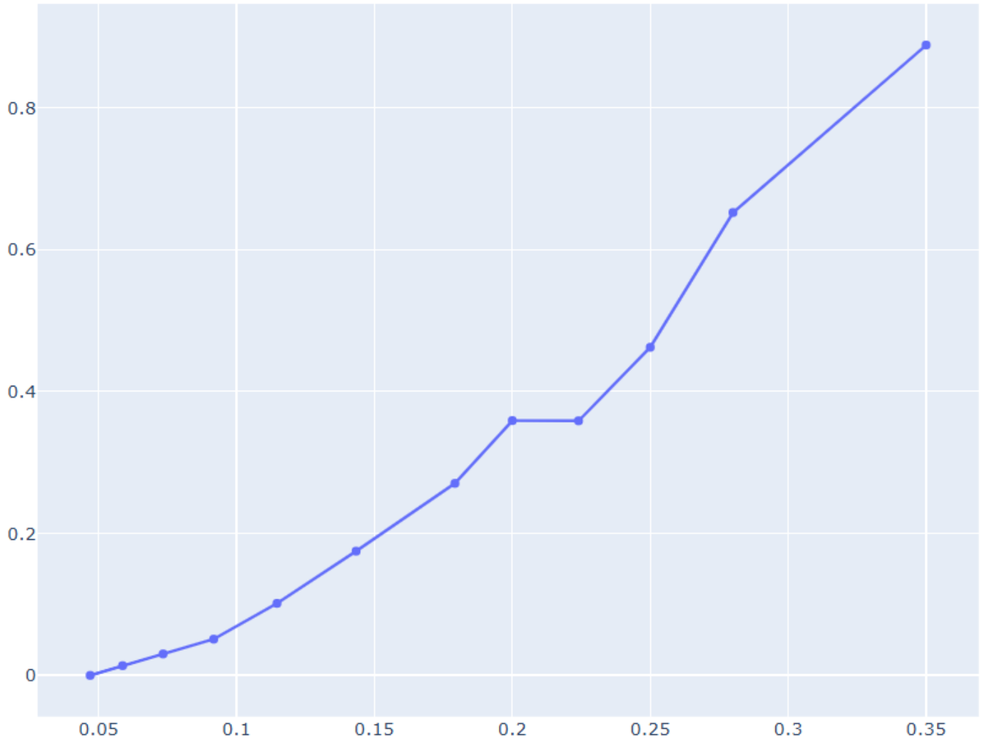
GHENT UNIVERSITY

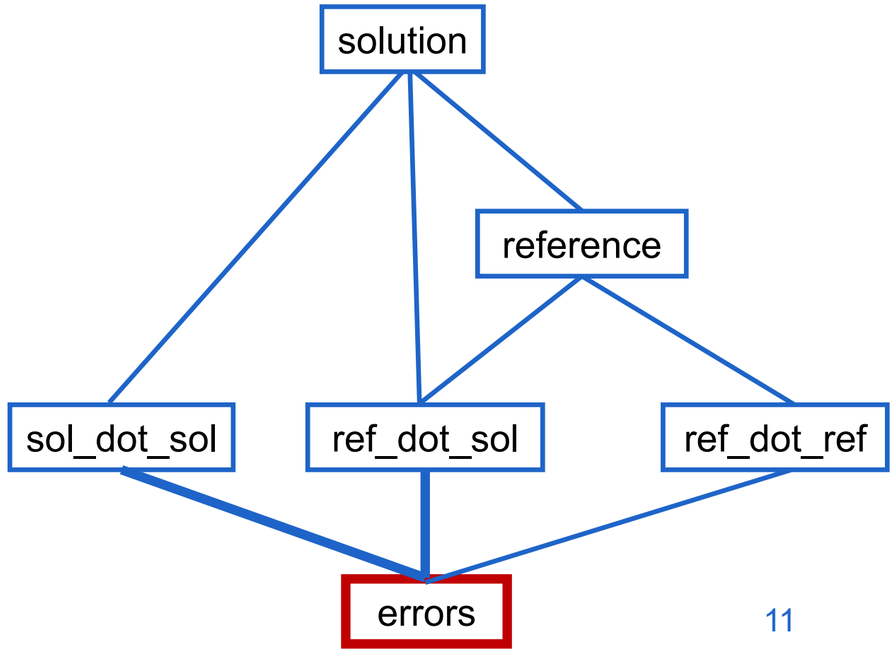
Example: convergence of the EFIE

- Compute the error (= distance) between solutions defined on different meshes:

$$\left|\left|a-b\right|\right|_S^2 = < a, Sa > \ -2Re < a, Sb > + < b, Sb >$$

- Challenge: the second term is based on a MoM matrix for basis and testing meshes that are not mutually conforming: special quadrature rules are required





GHENT UNIVERSITY

Example: the mixed formulation of the EFIE [Buffa et al]

- EFIE in terms of two scalar surface potentials (only for simply connected surface)
- Motivation:
  - Good LF properties
  - Amenable to domain decomposition techniques such as the Nitsche method and the local multi-trace method
  - Fully conformal time-domain discretization.
  - Efficient modelling of singular fields near edges and corners
- To avoid second order differential operators (and the necessity for C1 finite elements), Lagrange multipliers are introduced: use charge and scalar potential as auxiliary unknowns

$$
\begin{array}{rrr}
div\ A\ grad\ p + div\ A\ curl\ \lambda + & & div\ grad\ \phi = -div\ e_t^{inc} \\
curl\ A\ grad\ p + curl\ A\ curl\ \lambda & & = -curl\ e_t^{inc} \\
 & V\rho & -\phi = 0 \\
div\ grad\ p & -\rho & = 0
\end{array}
$$

- Is proven to be well-posed on $X = H^1 \times H^{\frac{1}{2}} \times H^{-\frac{1}{2}} \times H^1$.
- However, less symmetric than is appears
- Formulation operator is a compact perturbation of an elliptic operator if the first and last equation are interchanged: ruins the superficial symmetry!

GHENT UNIVERSITY

Example: the mixed formulation of the EFIE

– Resulting system:

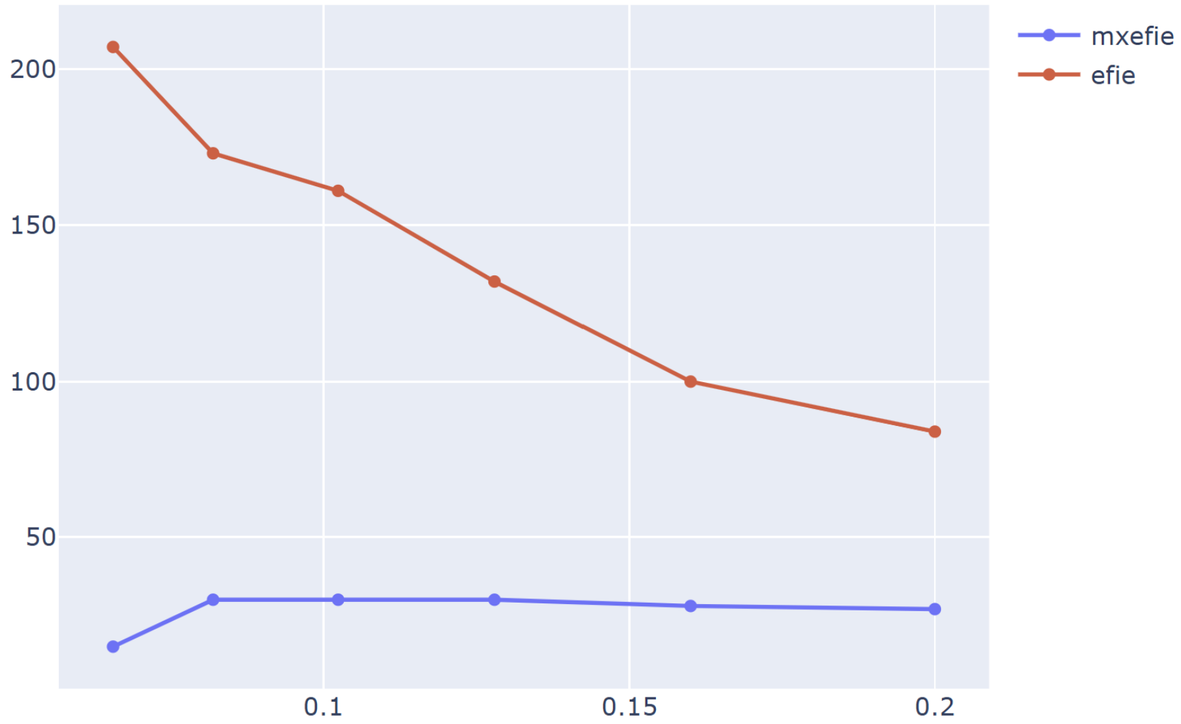
$$
\begin{aligned}
div\ grad\ p \qquad\qquad\qquad\qquad - \rho \qquad\qquad &= 0 \\
curl\ A\ grad\ p + curl\ A\ curl\ \lambda \qquad\qquad &= -curl\ e_t^{inc} \\
V\rho \qquad - \phi &= 0 \\
div\ A\ grad\ p + div\ A\ curl\ \lambda + \qquad div\ grad\ \phi &= -div\ e_t^{inc}
\end{aligned}
$$

– Even though less symmetric, this should be the starting point for building a Calderon preconditioner:

$$
\begin{pmatrix}
\Delta^{-1} & & & \\
& G^{-T}VG^{-1} & & \\
& & G^{-T}HG^{-1} & \\
& & & \Delta^{-1}
\end{pmatrix}
$$

GHENT UNIVERSITY

Example: Mixed EFIE




```
@target regularizer (geo, spaces; κ, h) -> begin
    @hilbertspace p q ρ φ
    @hilbertspace r s σ ψ

    (;X) = spaces
    Γ = geo.∂Ω[1]

    Id = BEAST.Identity()
    Δ = assemble(Id, gradient(X[r]), gradient(X[p]))
    Δ⁻¹ = BEAST.lu(Δ)

    V = Helmholtz3D.singlelayer(gamma=1.0)
    W = Helmholtz3D.hypersingular(gamma=1.0)

    Yq = BEAST.duallagrangecxd0(Γ)
    Yρ = BEAST.duallagrangec0d1(Γ)

    Iq = assemble(Id, X[q], Yq)
    Iρ = assemble(Id, X[ρ], Yρ)

    Dq = BEAST.GMRESSolver(Iq; verbose=false)
    Dρ = BEAST.GMRESSolver(Iρ; verbose=false)

    Vyy = assemble(V, Yq, Yq)
    Wyy = assemble(W, Yρ, Yρ)

    Rq = Dq' * Vyy * Dq
    Rρ = Dρ' * Wyy * Dρ

    (;R = assemble(Δ⁻¹[r,φ] + Δ⁻¹[ψ,p] + 4 *Rq[s,q] + 4 * Rρ[σ,ρ], X, X))
end
```

GHENT UNIVERSITY

(;R = assemble(Δ⁻¹[r,φ] + Δ⁻¹[ψ,p] + 4 *Rq[s,q] + 4 * Rρ[σ,ρ], X, X))

Example: PMCHWT

- Representation theorem holds in all domains:

$$\sum_{i=0}^{N} \langle (p_i, q_i), (-1/2 - A_i)(m_i, j_i) \rangle_\times =$$

$$\sum_{i=0}^{N} \left\langle (p_i, q_i), \left(e_i^{inc} \times n_i, n_i \times h_i^{inc}\right) \right\rangle_\times$$

- Restrict trial space to single trace functions:

$$m_i = n_i \times R_i g$$

with g a Nedelec function on the skeleton. All functions in Im R exhibit the correct continuity.

- Symmetrically, test only with such single trace (aka odd) functions:

$$\mathbf{R}_{\tilde{f}}{}^{f} \mathbf{A}_{ff} \mathbf{R}^{f}{}_{\tilde{f}} \mathbf{u}^{\tilde{f}} = \mathbf{R}_{\tilde{f}}{}^{f} \mathbf{e}_f$$

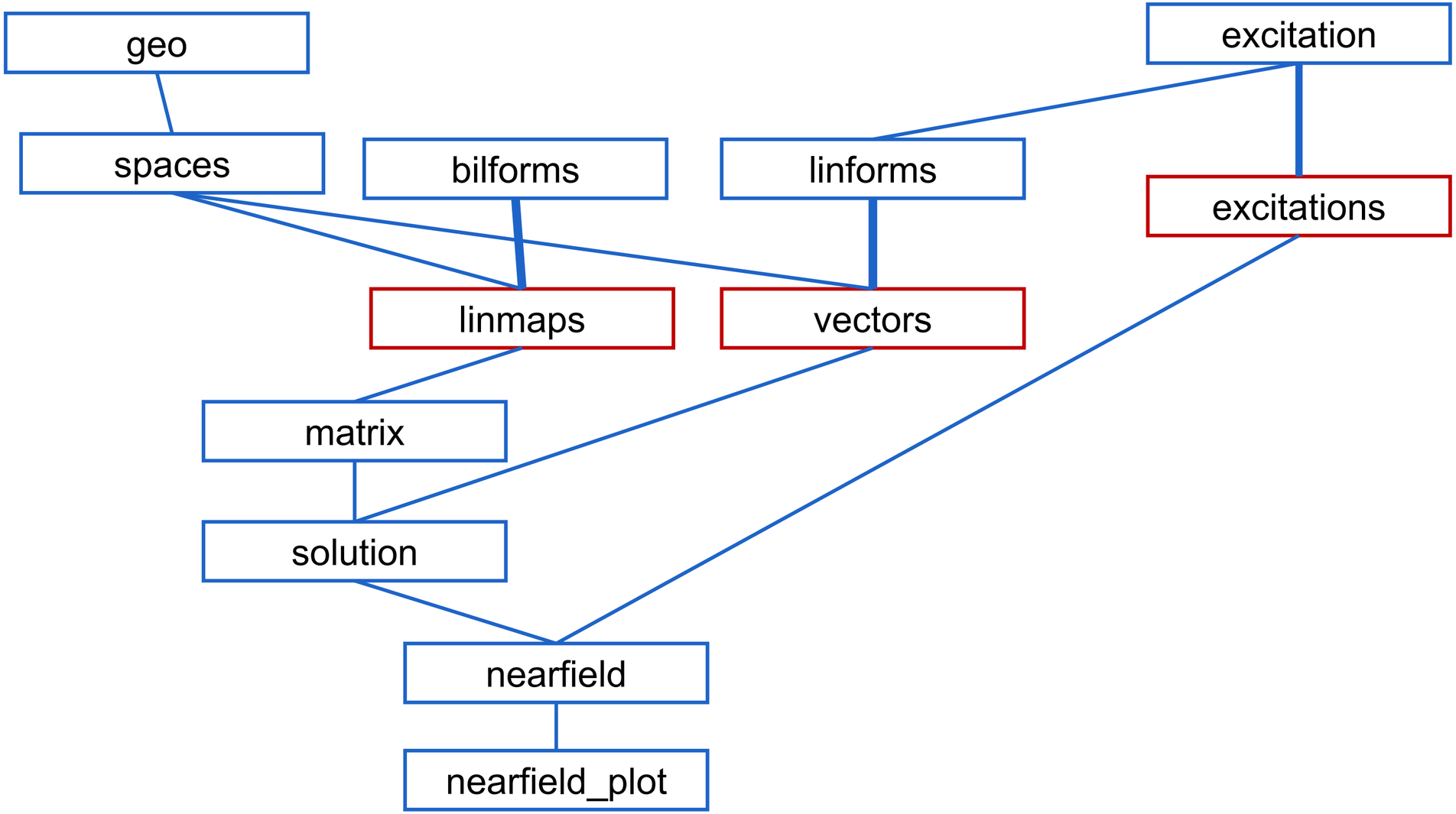
$$R^{ij}_{ee'} = \begin{cases} 0, & \text{if } e \text{ and } e' \text{ are disjoint} \\ +1, & \text{if } e \text{ and } e' \text{ have equal orientation} \\ -1, & \text{if } e \text{ and } e' \text{ have opposite orientation} \end{cases}$$

GHENT UNIVERSITY

```
36 @target formulation (geo, material, excitation) -> begin
37
38     numdoms = length(material)
39     (;∂Ω, Σ) = geo
40
41     (;κ, η) = first(material)
42     (;Einc, Hinc) = first(excitation)
43
44     e = (n × Einc) × n
45     h = (n × Hinc) × n
46
47     Ts = [Maxwell3D.singlelayer(wavenumber=m.κ) for m in material]
48     Ks = [Maxwell3D.doublelayer(wavenumber=m.κ) for m in material]
49
50     @hilbertspace m j
51     @hilbertspace p q
52     @hilbertspace z
53     u = BEAST.hilbertspace(:u, numdoms)
54     v = BEAST.hilbertspace(:v, numdoms)
55
56     b = -(e[p] - h[q])[v[1]]
57
58     Aloc = [
59         (-K)[p,m] + mat.η * T[p,j] -
60         (1/mat.η) * T[q,m] + (-K)[q,j] for (T,K,mat) in zip(Ts,Ks,material)
61     ]
62     A = sum(Aloc[i][u[i],v[i]] for i in 1:numdoms)
63
64     Edges = skeleton(Σ,1)
65     edges = [skeleton(∂Ωᵢ, 1) for ∂Ωᵢ in ∂Ω]
66     Rloc = [ CompScienceMeshes.embedding(edges[i], Edges) for i  in 1:numdoms]
67     R = sum((Rloc[i][m,p])[u[i],z] + (Rloc[i][j,q])[u[i],z] for i in 1:numdoms)
68
69     return (;bilforms=(;A,R), linforms=(;b))
70 end
```

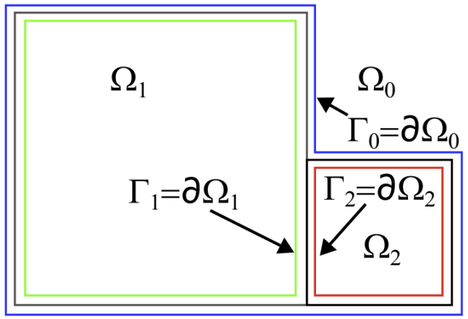
Example: fine grained splitting in targets

– If every domain is set up to have its own target, very efficient matrix recalculations can be achieved. Only those portions affected are recomputed:



GHENT UNIVERSITY

Examples: quasi-local multi-trace



- For the solution of the transmission problem, we have

$$\left(A_i - \frac{1}{2}\right) u_i = -u_i^{inc}$$

- Because of the continuity condition, we can replace the second term:

$$\left(\mathrm{A_i u_i} + \frac{1}{2}\sum_{j\neq i} I_{ij} u_j\right) = -u_i^{inc}$$

- Issues with this formulation:
  - Non-conforming: Convergence rates? Stability?
  - Not amenable to Calderon preconditioning
- Quasi-local multi-trace:

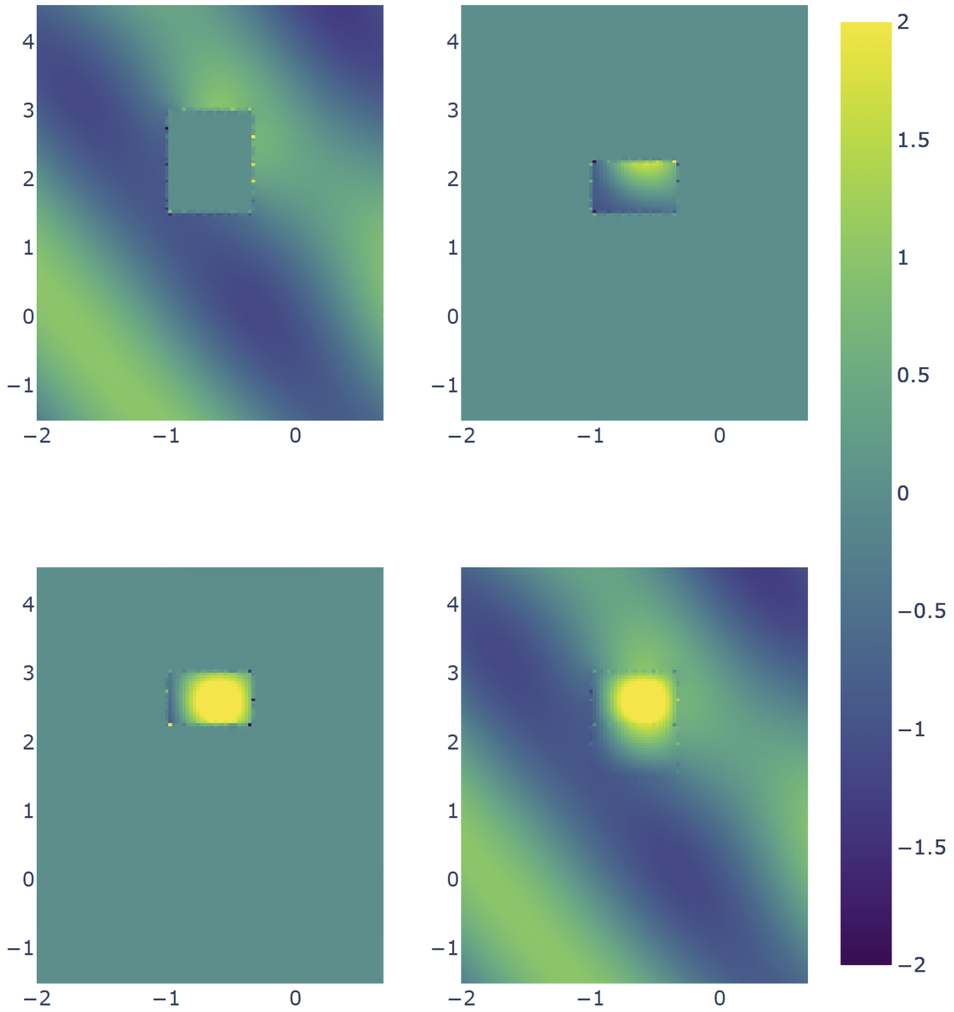
$$\left(A_i - \frac{1}{2} I_{ii}\right) u_i + \sum_j \left(K_{ij} + \frac{1}{2} I_{ij}\right) u_j = \cdots$$

$$(A_i + K_{ii}) u_i + \sum_{j\neq i} \left(K_{ij} + \frac{1}{2} I_{ij}\right) u_j = \cdots$$

- The second term is now the composition of a double layer potential and a trace operator, and so is continuous
- The system as a whole fulfills an inf-up condition and so is treatable by Calderon preconditioning

GHENT UNIVERSITY

Example: quasi-local multi-trace

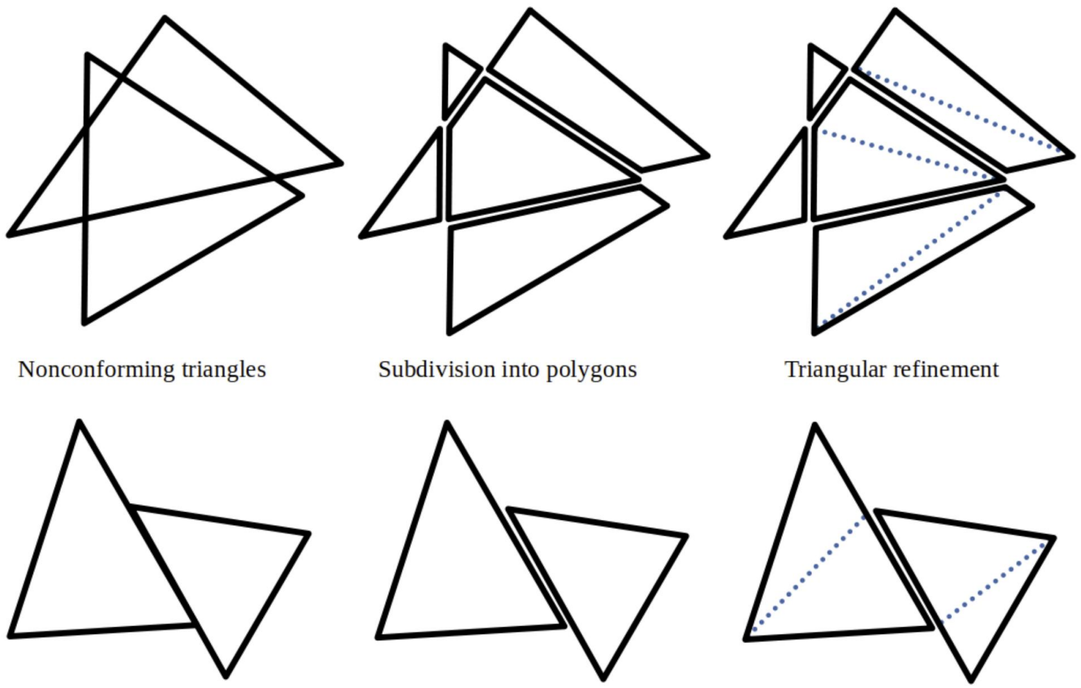


GHENT UNIVERSITY

```
@target formulation (material, excitation; h, κ, s) -> begin

    numdoms = length(material)

    @hilbertspace m j
    @hilbertspace k l
    p = BEAST.hilbertspace(:p, numdoms)
    q = BEAST.hilbertspace(:q, numdoms)

    (;Einc, Hinc) = excitation[1]
    e = (n × Einc) × n
    h = (n × Hinc) × n
    bloc = e[k] - h[l]
    b = bloc[q[1]]

    N = BEAST.NCross()
    T = [Maxwell3D.singlelayer(wavenumber=m.κ) for m ∈ material]
    K = [Maxwell3D.doublelayer(wavenumber=m.κ) for m ∈ material]
    K' = Maxwell3D.doublelayer(wavenumber=κ*im)

    Aloc = [Kᵢ[k,m] - Tᵢ[k,j] + Tᵢ[l,m] + Kᵢ[l,j] for (Tᵢ,Kᵢ) ∈ zip(T,K)]
    A'loc = K'[k,m] + K'[l,j]

    A = sum(Aloc[i][q[i],p[i]] for i in 1:numdoms)
    A += sum(A'loc[q[i],p[i]] for i in 1:numdoms)

    Jloc = -0.5*N[k,m] - 0.5*N[l,j]
    J = sum(Jloc[q[i],p[j]] for i in 1:numdoms for j in 1:numdoms if i != j)
    J -= sum(A'loc[q[i],p[j]] for i in 1:numdoms for j in 1:numdoms if i != j)
    M = A - J

    Rloc = [-Tᵢ[k,j] + Tᵢ[l,m] for (Tᵢ,Kᵢ) ∈ zip(T,K)]
    R = sum(Rloc[i][q[i],p[i]] for i in 1:numdoms)

    Dloc = N[k,m] + N[l,j]
    D = sum(Dloc[q[i],p[i]] for i in 1:numdoms)

    return (;bilforms=(;A, J, M, R, D), linforms=(;b))
end
```

Example: Quasi-local multi-trace

- Note: Neighboring domains can be meshed independently.
- Accurate integration requires:
  - ad-hoc generation of mutually conforming refinements
  - expression of restrictions of basis functions in terms of functions on the refinement
  - calculation of interactions on the refinements and adding contributions



GHENT UNIVERSITY

Example: Quasi-local multi-trace

- BEAST is built around the concepts of quadrature strategy and quadrature rules:
  - The strategy determines which rule is chosen given a relative positioning of panels
  - For each rule there is an overload of the function computing the local interactions

```
struct NonConformingIntegralOpQStrat{S} <: AbstractQuadStrat
    conforming_qstrat::S
end

function quaddata(a, X, Y, tels, bels, qs::NonConformingIntegralOpQStrat)
    return quaddata(a, X, Y, tels, bels, qs.conforming_qstrat)
end

function quadrule(a, 𝑥, 𝑦, i, τ, j, σ, qd,
    qs::NonConformingIntegralOpQStrat)

    if CompScienceMeshes.overlap(τ, σ)
        return NonConformingOverlapQRule(qs.conforming_qstrat)
    end

    for (i,λ) in pairs(faces(τ))
        for (j,μ) in pairs(faces(σ))
            if CompScienceMeshes.overlap(λ, μ)
                return NonConformingTouchQRule(qs.conforming_qstrat, i, j)
    end end end

    # Either positive distance or common vertex, both can
    # be handled directly by the parent quadrature strategy
    return quadrule(a, 𝑥, 𝑦, i, τ, j, σ, qd, qs.conforming_qstrat)
end
```

GHENT UNIVERSITY

Example: Quasi-local multi-trace

- BEAST is built around the concepts of quadrature strategy and quadrature rules:
  - The strategy determines which rule is chosen given a relative positioning of panels
  - For each rule there is an overload of the function computing the local interactions

```
function momintegrals!(op, test_local_space, basis_local_space,
    test_chart::CompScienceMeshes.Simplex, basis_chart::CompScienceMeshes.Simplex,
    out, qrule::NonConformingOverlapQRule)

    num_tshapes = numfunctions(test_local_space, domain(test_chart))
    num_bshapes = numfunctions(basis_local_space, domain(basis_chart))

    test_charts, tclps = CompScienceMeshes.intersection_keep_clippings(test_chart, basis_chart)
    _,           bclps = CompScienceMeshes.intersection_keep_clippings(basis_chart, test_chart)
    bsis_charts = copy(test_charts)

    for tclp in tclps append!(test_charts, tclp) end
    for bclp in bclps append!(bsis_charts, bclp) end

    T, h = coordtype(test_chart), max(volume(test_chart), volume(test_chart))
    test_charts = [ch for ch in test_charts if volume(ch) .> 1e6 * eps(T) * h]
    bsis_charts = [ch for ch in bsis_charts if volume(ch) .> 1e6 * eps(T) * h]
    isempty(test_charts) || isempty(bsis_charts) && return

    test_overlaps = map(test_charts) do tchart
        simplex(
            carttobary(test_chart, tchart.vertices[1]),
            carttobary(test_chart, tchart.vertices[2]),
            carttobary(test_chart, tchart.vertices[3]))
    end

    trial_overlaps = map(bsis_charts) do bchart
        simplex(
            carttobary(basis_chart, bchart.vertices[1]),
            carttobary(basis_chart, bchart.vertices[2]),
            carttobary(basis_chart, bchart.vertices[3]))
    end
    qstrat = CommonFaceOverlappingEdgeQStrat(qrule.conforming_qstrat)
    qdata = quaddata(op, test_local_space, basis_local_space,
        test_charts, bsis_charts, qstrat)

    zlocal = zero(out)
    P = zeros(T, num_tshapes, num_tshapes)
    Q = zeros(T, num_bshapes, num_bshapes)
    for (p,tchart) in enumerate(test_charts)
        restrict!(P, test_local_space, test_chart, tchart, test_overlaps[p])
        for (q,bchart) in enumerate(bsis_charts)
            restrict!(Q, basis_local_space, basis_chart, bchart, trial_overlaps[q])

            qrule = quadrule(op, test_local_space, basis_local_space,
                p, tchart, q, bchart, qdata, qstrat)

            fill!(zlocal, 0)
            momintegrals!(op, test_local_space, basis_local_space,
                tchart, bchart, zlocal, qrule)

            for i in axes(P,1) for j in axes(Q,1)
                    for k in axes(P,2) for l in axes(Q,2)
                        out[i,j] += P[i,k] * zlocal[k,l] * Q[j,l]
end end end end end end end
```