DYALOG

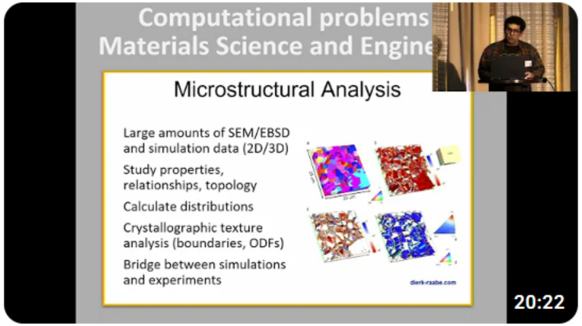
Elsinore 2023

APL and Metallurgy

Jesús Galán López (Spain)

Dyalog / Ghent University

GHENT UNIVERSITY



Dyalog and Academia // Jesús Galán López // Dyalog '22

232 views • 10 months ago

Dyalog Usermeeting

Jesús Galán López Jesús and Gitte introduce the plan for bringing **Dyalog** back into **academia**. The final objective is to position ...

Introduction and background | STEM in academia and where APL fits | Project goals | Action plan |... 9 chapters

Dyalog and Academia

- Get visibility in academic and research environments
- Learn about them and from them
- Study and improve the use of APL as a tool for education and research of technical topics
- Introduce a new generation to APL
- Establish fruitful relationships with the academic world

APL and Metallurgy

- Use APL to teach / learn about complex topics
  - Mathematical methods
  - Crystallography and crystal plasticity
  - Modelling of thermo-mechanical processes
- Evaluate capacities of APL as a teaching tool when compared with mainstream languages
- More presence in the classroom and the student community

GHENT UNIVERSITY

APL and Metallurgy

1. Initial plan
2. Results
3. Conclusions
4. Next

The plan

- Evaluation of state of the art
- Development: code, documentation and learning materials
- Research: code and scientific publications
- Dissemination of APL (evangelism)

WP1	1.1 Current usage of software and programming languages at the university
	1.2 Evaluation of current capacities of APL as a teaching and research tool
WP2	2.1 Introduction to APL for material science and engineering students
	2.2 Data analysis: IO, basic processing, visualisation
	2.3 Algebra, statistics and calculus
	2.4 Advanced techniques for metallurgical research
WP3	3.1 Microstructural analysis
	3.2 Crystallographic texture / crystal plasticity / transformation models
WP4	4.1 Documentation and tutorials
	4.2 Scientific articles and conferences
	4.3 Promotion of APL

What happened

- Kept goals
- Merged some tasks, split some others
- Amount of work and time sometimes different to expectations
- Results

The plan

- Evaluation of state of the art
- Development: code, documentation and learning materials
- Research: code and scientific publications
- Dissemination of APL (evangelism)

WP1	1.1 Current usage of software and programming languages at the university
	1.2 Evaluation of current capacities of APL as a teaching and research tool
WP2	2.1 Introduction to APL for material science and engineering students
	2.2 Data analysis: IO, basic processing, visualisation
	2.3 Algebra, statistics and calculus
	2.4 Advanced techniques for metallurgical research
WP3	3.1 Microstructural analysis
	3.2 Crystallographic texture / crystal plasticity / transformation models
WP4	4.1 Documentation and tutorials
	4.2 Scientific articles and conferences
	4.3 Promotion of APL

Results

- Evaluation of state of the art
- Development: code, documentation and learning materials
- Research: code and scientific publications
- Dissemination of APL (evangelism)

WP1	1.1 Current usage of software and programming languages at the university
	1.2 Evaluation of current capacities of APL as a teaching and research tool
WP2	2.1 Introduction to APL for material science and engineering students
	2.2 Data analysis: IO, basic processing, visualisation
	2.3 Algebra, statistics and calculus
	2.4 Advanced techniques for metallurgical research
WP3	3.1 Microstructural analysis
	3.2 Crystallographic texture / crystal plasticity / transformation models
WP4	4.1 Documentation and tutorials
	4.2 Scientific articles and conferences
	4.3 Promotion of APL

Evaluation of state of the art

- Programming languages and software at the university
  - Review current literature
  - Personal point of view and real cases
  - Programming languages strengths and weaknesses
  - Report

Evaluation of state of the art

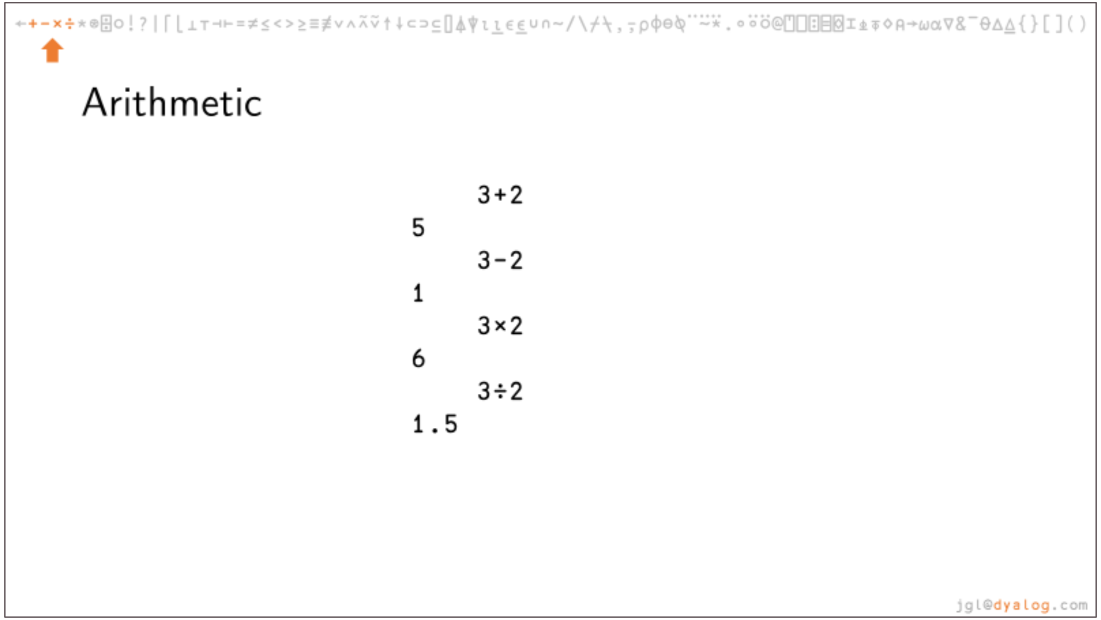
- APL at the faculty
  - Unheard of
  - Looks hard (those symbols!)
  - How do I … in APL?
  - How does APL compare to …?
  - Can APL solve *my* problem?

Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides

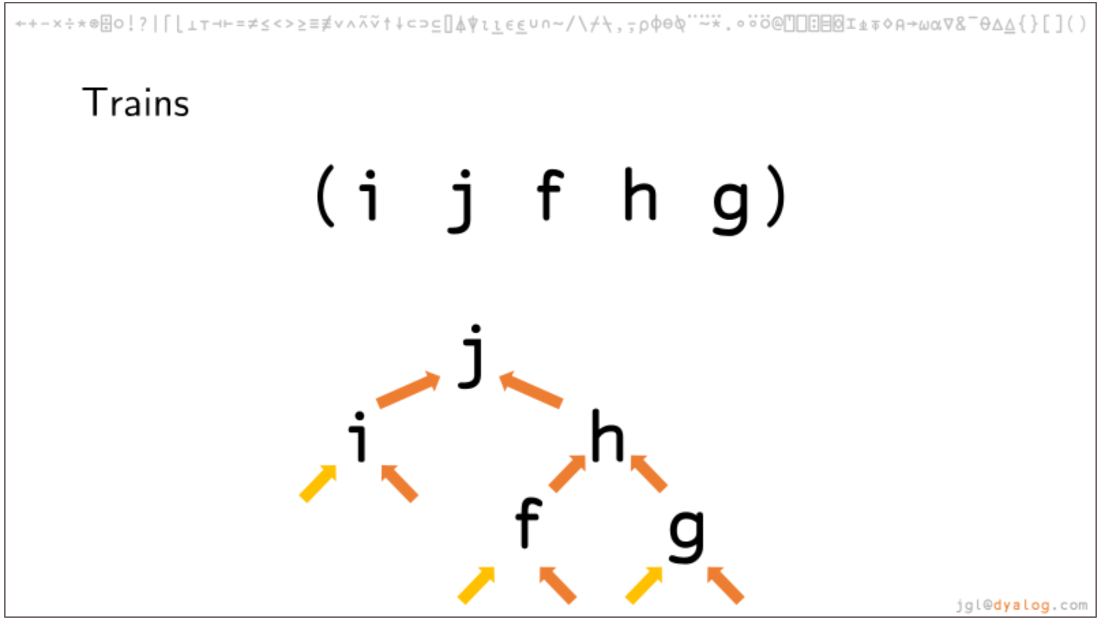
Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



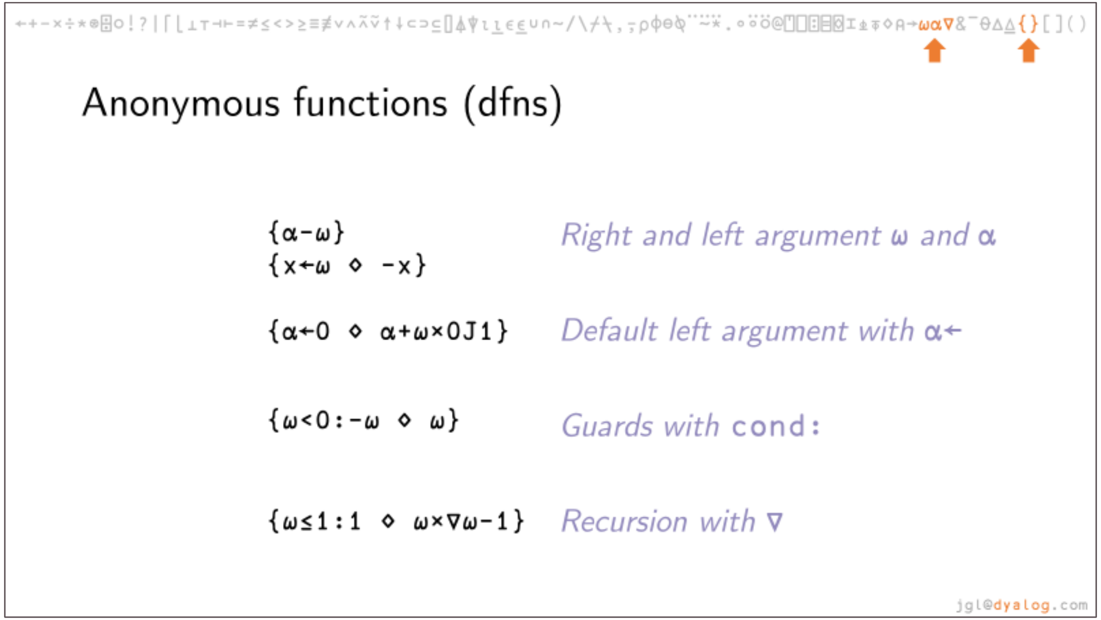
Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides

←+-×÷*⍟⌹○!?|⌈⌊⊥⊤⊣⊢=≠≤<>≥≡≢∨∧⍲⍱↑↓⊂⊃⊆⌷⍋⍒⍳⍸∊⍷∪∩~/\⌿⍀,⍪⍴⌽⊖⍉¨⍨⍣.∘⍤⍥@⎕⍠⌸⌺⌶⍎⍕⋄⍝→⍵⍺∇&¯⍬∆⍙{}[]()

System

Primitives
⎕CT ⎕DCT ⎕DIV ⎕FR ⎕IO ⎕ML ⎕PP ⎕RL

Session
⎕AI ⎕AN ⎕CLEAR ⎕CY ⎕DL ⎕LOAD ⎕OFF ⎕PATH ⎕SAVE ⎕TS

Constants
⎕A ⎕D ⎕NULL

Tools
⎕C ⎕CMD ⎕CSV ⎕DR ⎕DT ⎕FMT ⎕JSON ⎕MAP ⎕NA ⎕R ⎕S ⎕SH ⎕UCS ⎕USING ⎕VFI ⎕XML

Functions and Operators
⎕AT ⎕CR ⎕ED ⎕EX ⎕FX ⎕LOCK ⎕MONITOR ⎕OR ⎕NR ⎕PROFILE ⎕REFS ⎕STOP ⎕TRACE ⎕VR

Namespaces and Objects
⎕BASE ⎕CLASS ⎕CS ⎕DF ⎕FIX ⎕INSTANCES ⎕NEW ⎕NS ⎕SRC ⎕THIS

Input/Output
⎕ ⍞ ⎕ARBIN ⎕ARBOUT ⎕RTL

Component Files
⎕FAPPEND ⎕FAVAIL ⎕FCHK ⎕FCOPY ⎕FCREATE ⎕FDROP ⎕FERASE ⎕FHIST ⎕FHOLD ⎕FLIB ⎕FNAMES ⎕FNUMS ⎕FPROPS ⎕FRDAC ⎕FRDCI ⎕FREAD ⎕FRENAME ⎕FREPLACE ⎕FRESIZE ⎕FSIZE ⎕FSTAC ⎕FSTIE ⎕FTIE ⎕FUNTIE

Native Files
⎕MKDIR ⎕NAPPEND ⎕NCOPY ⎕NCREATE ⎕NDELETE ⎕NERASE ⎕NEXISTS ⎕NGET ⎕NINFO ⎕NLOCK ⎕NMOVE ⎕NNAMES ⎕NNUMS ⎕NPARTS ⎕NPUT ⎕NREAD ⎕NRENAME ⎕NREPLACE ⎕NRESIZE ⎕NSIZE ⎕NTIE ⎕NUNTIE ⎕NXLATE

Threads
⎕TCNUMS ⎕TID ⎕TKILL ⎕TNAME ⎕TNUMS ⎕TSYNC

Synchronisation
⎕TGET ⎕TKILL ⎕TPOOL ⎕TPUT ⎕TREQ

Error Handling
⎕DM ⎕DMX ⎕EM ⎕EN ⎕EXCEPTION ⎕SIGNAL ⎕TRAP

Workspace
⎕LC ⎕LX ⎕NC ⎕NL ⎕NSI ⎕RSI ⎕SI ⎕SHADOW ⎕SIZE ⎕STACK ⎕STATE ⎕WA ⎕WSID ⎕XSI

Shared Variables
⎕SVC ⎕SVO ⎕SVQ ⎕SVR ⎕SVS

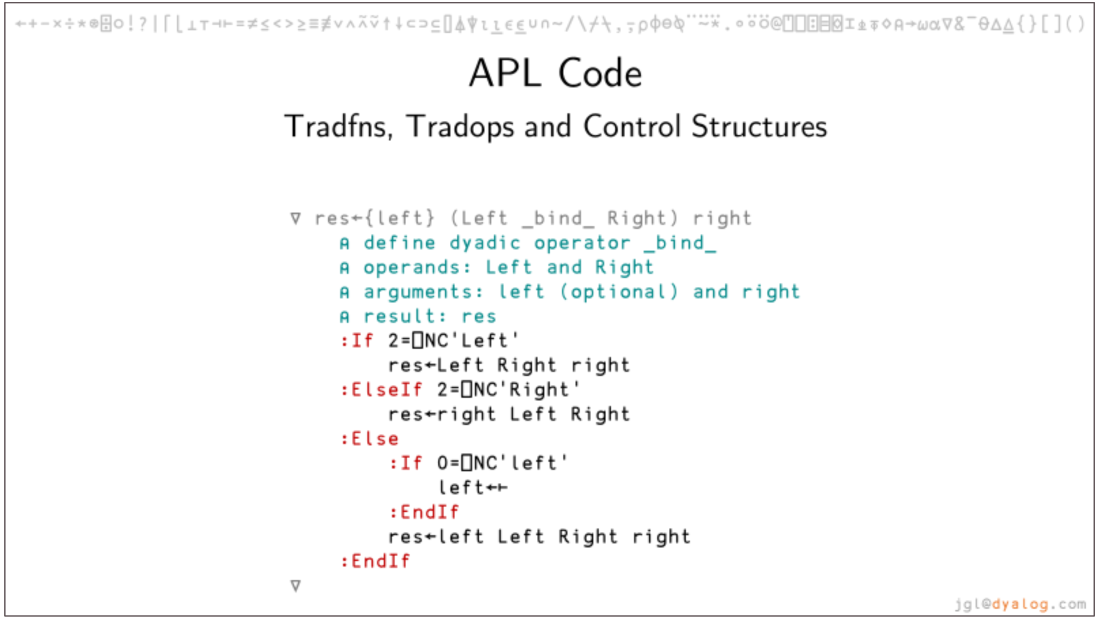
GUI and COM
⎕DQ ⎕EXPORT ⎕NQ ⎕WC ⎕WG ⎕WN ⎕WS ⎕WX

Misc
⎕Á ⎕AV ⎕AVU ⎕KL ⎕PFKEY ⎕SD ⎕SM ⎕SR ⎕OPT ⎕TC ⎕XT

jgl@dyalog.com

Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



APL Code

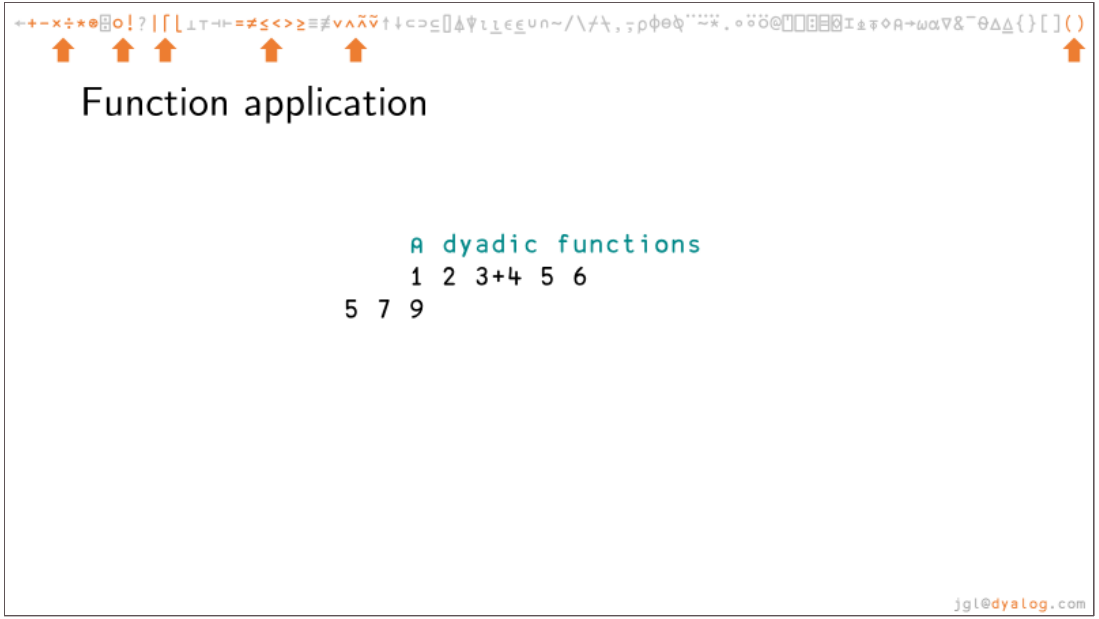
Tradfns, Tradops and Control Structures

```
∇ res←{left} (Left _bind_ Right) right
    ⍝ define dyadic operator _bind_
    ⍝ operands: Left and Right
    ⍝ arguments: left (optional) and right
    ⍝ result: res
    :If 2=⎕NC'Left'
        res←Left Right right
    :ElseIf 2=⎕NC'Right'
        res←right Left Right
    :Else
        :If 0=⎕NC'left'
            left←⊢
        :EndIf
        res←left Left Right right
    :EndIf
∇
```

jgl@dyalog.com

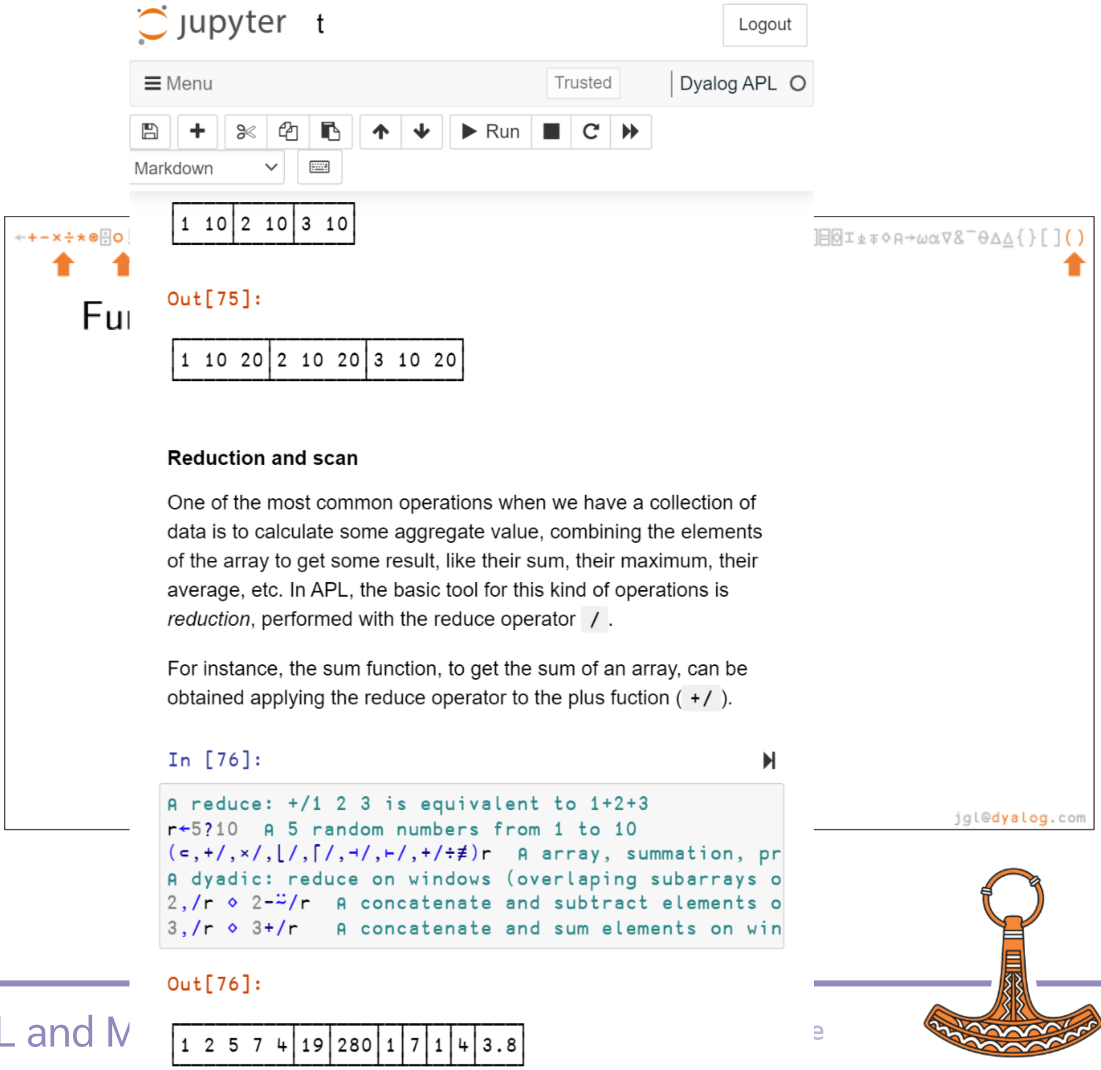
Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



Development

- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides



Development

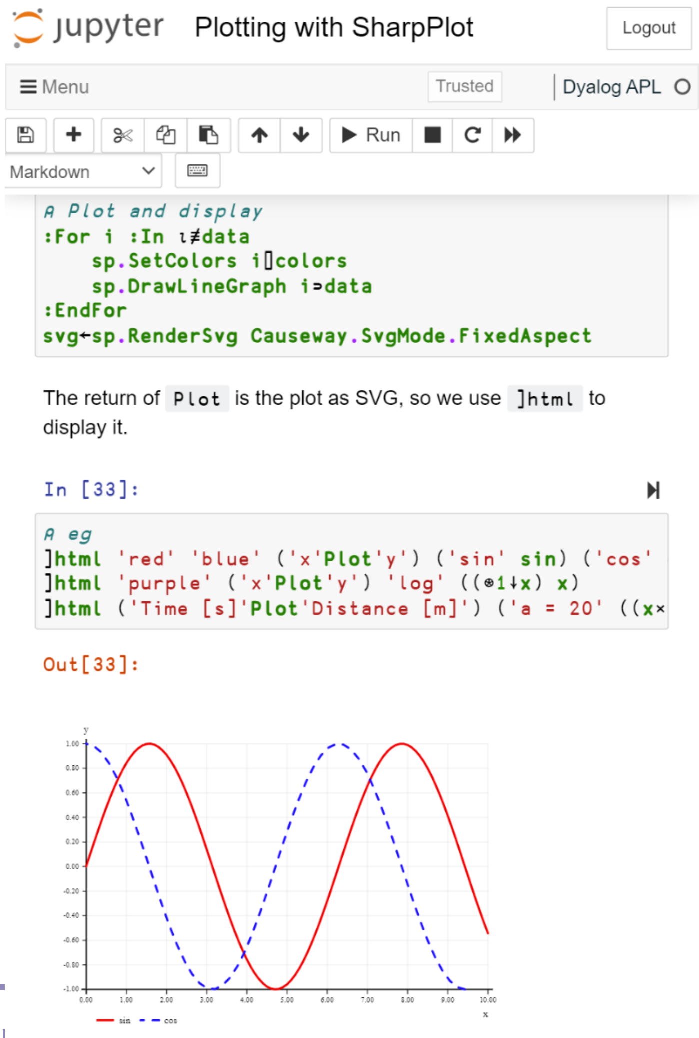
- Introduction tutorial
  - No previous programming experience
  - Cover: syntax, primitives, dfns, system
  - Tradfns, namespaces, classes only mentioned
  - Simple examples
  - Jupyter notebook and slides

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)



Development

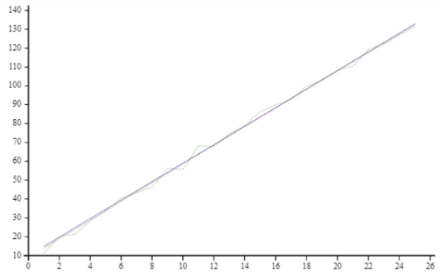
- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)

jupyter Linear Fitting Logout

Menu Trusted Dyalog APL

Run

Markdown



R-squared value

We still need to determine the R-squared value, which can be calculated as one minus the ratio of the sums of squares (`÷⍥(+/×⍨)`) of differences between the real and estimated values and between the real values and their mean:

In [8]:

```
1-(y-a+b×x)÷⍥(+/×⍨)(⊢-+⌿÷≢)y   ⍝ R-squared value
```

Out[8]:

```
0.9971435494
```

The R-squared value is defined not only for linear fittings, but also for any other fitting we perform on some data. Therefore, we can define an operator that takes the fitted function as operand (in our

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)

jupyter Calculus

Logout

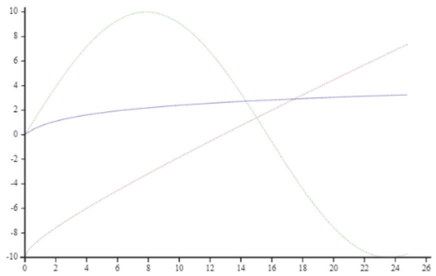
Menu | Trusted | Dyalog APL

Run

Code

```
f←⍟1+x ⋄ g←10-⍨(x*0.5)+x÷2 ⋄ h←10×1○x÷5
]plot (f g h h)x
```

Out[1]:



Derivative

The derivative will be calculated as a simple ratio of the difference between two consequtive values. The differences will be calculated using the n-wise reduction operator (dyadic `/`).

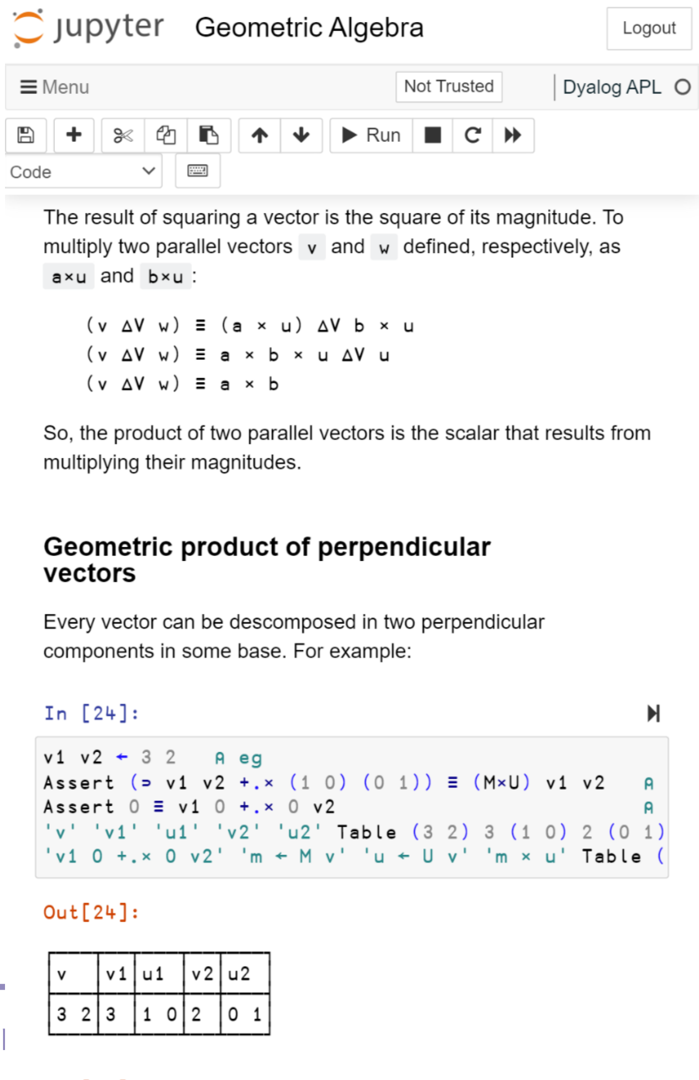
In [2]:

```
dy←2-/f ⋄ dx←2-/x ⋄ d←dy÷dx
]plot d x
```

Out[2]:

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)



The result of squaring a vector is the square of its magnitude. To multiply two parallel vectors `v` and `w` defined, respectively, as `a×u` and `b×u`:

```
(v ΔV w) ≡ (a × u) ΔV b × u
(v ΔV w) ≡ a × b × u ΔV u
(v ΔV w) ≡ a × b
```

So, the product of two parallel vectors is the scalar that results from multiplying their magnitudes.

Geometric product of perpendicular vectors

Every vector can be descomposed in two perpendicular components in some base. For example:

In [24]:

```
v1 v2 ← 3 2   ⍝ eg
Assert (⊃ v1 v2 +.× (1 0) (0 1)) ≡ (M×U) v1 v2   ⍝
Assert 0 ≡ v1 0 +.× 0 v2                          ⍝
'v' 'v1' 'u1' 'v2' 'u2' Table (3 2) 3 (1 0) 2 (0 1)
'v1 0 +.× 0 v2' 'm ← M v' 'u ← U v' 'm × u' Table (
```

Out[24]:

v	v1	u1	v2	u2
3 2	3	1 0	2	0 1

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)

```
Rectangle.Area ← ×
   Square.Area ← ×⍨

⍝ uniform acceleration
t⊥(a÷2) v0 x0
```

jupyter formulae

Logout

Menu | Not Trusted | Dyalog APL

Run | Code

Geometry

In [16]:

```
:Namespace Geometry
    ⍝ 2D
    Hypotenuse ← (÷2)*⍨+⍥(×⍨)  ⍝ catheti
    :Namespace Rectangle
        Area ← ×                    ⍝ sides
        Perimeter ← 2×+             ⍝ sides
        Diagonal ← ##.Hypotenuse    ⍝ sides
    :EndNamespace
    :Namespace Square
        Area ← ##.Rectangle.Area⍨           ⍝ side
        Perimeter ← ##.Rectangle.Perimeter⍨ ⍝ side
        Diagonal ← ##.Rectangle.Diagonal⍨   ⍝ side
    :EndNamespace
    :Namespace Triangle
        Area ← 2÷⍨×      ⍝ base height
        Perimeter ← +/   ⍝ sides
    :EndNamespace
    :Namespace Rhombus
        ⍝ diagonals
        Area ← 2÷⍨×                  ⍝ diagonals
        Side ← 2÷⍨##.Hypotenuse      ⍝ diagonals
        Perimeter ← 4×Side           ⍝ diagonals
    :EndNamespace
    :Namespace Trapezoid
        Area ← 2÷⍨×∘+/⍨  ⍝ height bases
        Median ← 2÷⍨+/   ⍝ bases
        Perimeter ← +/   ⍝ sides
    :EndNamespace
    :Namespace Circle
        Diameter ← 2∘×       ⍝ radious
        Perimeter ← (○2)∘×   ⍝ radious
```

Development

- Specialized tutorials: engineering (jupyter)
  - Plotting
  - Linear fitting
  - Simple calculus
  - Geometric algebra
  - Formulae (WIP)

Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

Development

- Specialized tutorials: materials (jupyt
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

Introduction

This notebook presents a short tutorial on how to use Dyalog APL for the analysis of tensile experimental data. The tutorial is directed towards scientists, engineers and students who have already learnt the basics of the APL language and intend to use Dyalog APL for their data analysis work.

The topic of tensile data analysis is chosen because it represents a well known task for many engineers of different disciplines, and it is a perfect example of the more general procedure of reading data from a file, doing some basic processing, and plotting the obtained results, which is familiar to many engineering researchers and students.

Tensile analysis

Tensile testing is one of the most common experimental methods for the determination of the mechanical properties of materials. Materials scientists employ tensile tests to find different properties that define the mechanical behaviour of materials, and mechanical, aeronautical, and civil engineers, use these properties for the design of structures.

In a uniaxial tensile test (the most common kind of tensile test), a specimen with a predefined geometry is subjected to a uniaxial load that deforms the material under controlled conditions. The force during the experiment is measured with a load cell, while the deformation on the sample is measured either directly on the specimen using an extensometer or a strain gauge, or indirectly based on the crosshead displacement of the testing machine. From this data, strain and stress are calculated, respectively, as displacement with respect to the original length and force divided by cross sectional area (width multiplied by thickness):

$$e_{eng} = \frac{\Delta l}{l_0} = \frac{l - l_0}{l_0} \qquad S_{eng} = \frac{F}{A} = \frac{F}{w \cdot t}$$

It is customary to present this data as a tensile diagram or strain-stress curve.

During a tensile experiment, both the length and the area of the specimen change. When strain and stress are calculated with respect to the initial values, as they are in the formulas presented above, we will call them *engineering strain* and *engineering stress*. When they are calculated with respect to the instantaneous values, we will call them *true strain* and *true stress*. Since true values are not dependent on initial conditions, true values are not dependent on a specific specimen geometry, at difference of the corresponding engineering values. Assuming that the volume is conserved, true and engineering strain and stress will be correlated by the equations:

$$e_{true} = \ln(1 + e_{eng}) \quad (1)$$

$$S_{true} = S_{eng}(1 + e_{eng}) \quad (2)$$

Development

- Specialized tutorials: materials (jupyt
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

¯0.000397524	4.758206844

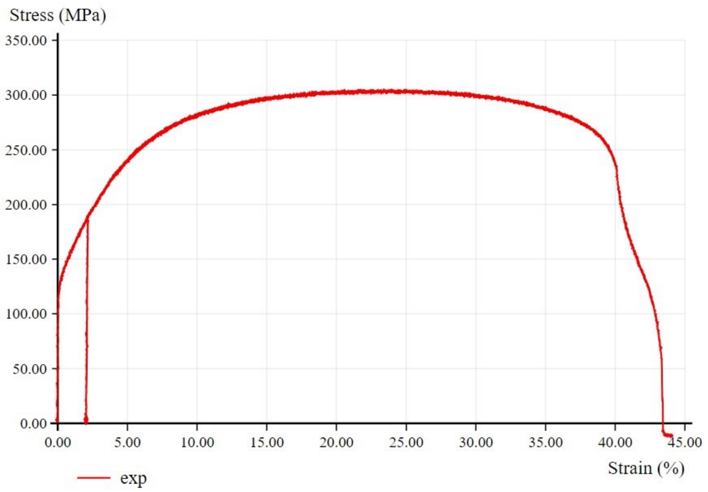
We will store the strain-stress data as a vector with two columns. Since it is a trivial operation, we will not define specific functions to extract the strain and stress data, and will instead use `⊃` for strain and `⊃⌽` for stress.

Data files might be quite large, and there could be a significant amount of experimental noise:

In [10]:
```
'Number of rows:',≢⊃e
]html PlotTD 'exp' e
```

Out[10]: Number of rows: 20040

Out[10]:



We observe that the data does not only contain a very large number of points and experimental scatter. Other issues are that it does not exactly start at zero, there are some suspicious points around 2% strain which are assumed to be erroneous and need to be discarded, and the measurement continues after fracture of the sample, with data that is not relevant.

Young modulus and initial strain offset

In order to find the Young modulus, we need to perform a linear fitting. The curve should start at strain zero,

Development

- Specialized tutorials: materials (jupyt
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

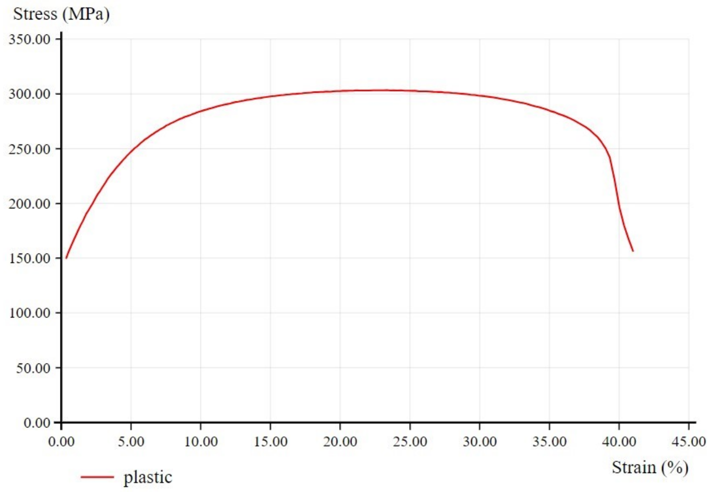
(x÷y).

The `Plastic` operator allows to specify what is the yielding offset as `αα` (usually defined as the point of 0.2% or 0.02% of plastic strain), then uses `Clean` to remove the points with plastic strain higher than specified.

In [18]:

```
Plastic ← {x y←α ⋄ αα≤∘⊃Clean ω-(y÷⍨x+⊃⌽ω)0}   ⍝ get plastic curve from total tensile curv
⍝ plastic curve with 0.2% yield offset
p←x y(0.002 Plastic)c
]html PlotTD 'plastic' p
```

Out[18]:



True tensile curve, hardening rate, and UTS point

After calculating the plastic strain, our next goal is to calculate the true curve and the strain hardening rate. The true strain can be calculated from the engineering strain as `⍟1+⊣`. To calculate the true stress, we need the engineering stress and the engineering strain. If the engineering strain is the left argument and the engineering stress the right one, it is calculated as `⊢×1+⊣`. The strain hardening rate is calculated as the derivative of the true plastic curve using `D`, from the Calculus notebook. The UTS point is the point where the true curve and the strain hardening rate intercept.

The `TrueRate` function is used to calculate both the true curve until the UTS point (the uniform deformation

Development

- Specialized tutorials: materials (jupyt
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

(x÷y).

The `Plastic` operator allows to specify what is the yielding offset as `αα` (usually defined as the point of 0.2% or 0.02% of plastic strain), then uses `Clean` to remove the points with plastic strain higher than specified.

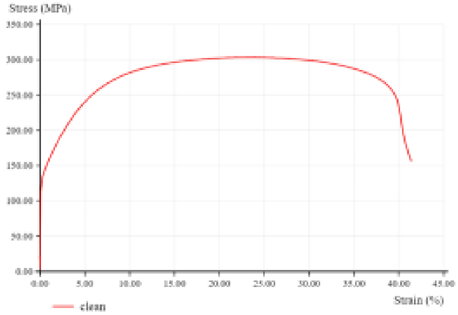
In [18]:
```
Plastic ← {x y←α ⋄ αα≤∘⊃Clean ω-(y÷⍨x+⊃⌽ω)0}   ⍝ get plastic curve from total tensile curv
⍝ plastic curve with 0.2% yield offset
p←x y(0.002 Plastic)c
]html PlotTD 'plastic' p
```

Out[18]:

Stress (MPa)
350.00

Resampling and cleaning

```
Resample ← {(α-1){(⊃ω),÷∘n+/ωρ⍨α,n←⌊α÷⍨≢ω}¨ω}        ⍝ average rows
Smooth ← {α←3 ⋄ ωResample⍨⌈α÷⍨≢⊃ω}                  ⍝ average rows
Clean ← {α←⊢ ⋄ (α αα ω)∘/¨⎕¨⍨∘(⊂⍤2⍋∘⊃)⍨ω/¨⍨⊂≠⊃ω}  ⍝ select points
c←((0.02>⊃)∨150<⊃∘⌽)Clean 250 Resample e ⍝ strain<0.02 or stress>150
]html PlotTD 'clean' c
```



jgl@dyalog.com
jesus.galanlopez@ugent.be

true strain can be calculated from the engineering strain as `⍟1+⊣`. To calculate the true stress, we need the engineering stress and the engineering strain. If the engineering strain is the left argument and the engineering stress the right one, it is calculated as `⊢×1+⊣`. The strain hardening rate is calculated as the derivative of the true plastic curve using `D`, from the Calculus notebook. The UTS point is the point where the true curve and the strain hardening rate intercept.

The `TrueRate` function is used to calculate both the true curve until the UTS point (the uniform deformation

jgl@dyalog.c

Development

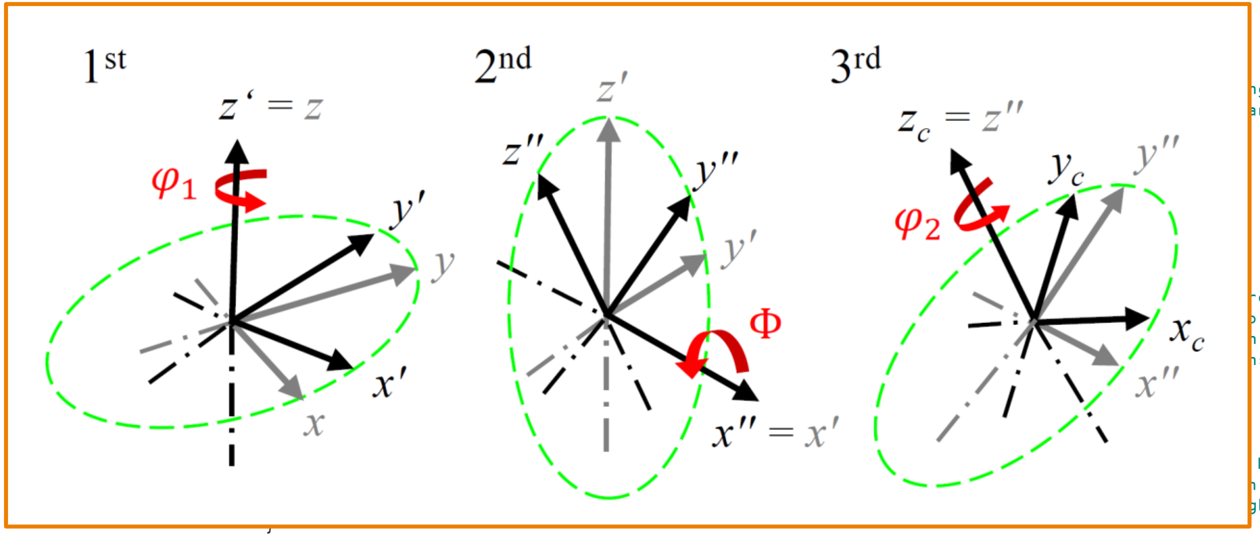
- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)                                ⍝ product components ⎕IO=0
    u ← ↑(1 ¯1 ¯1 ¯1)(1 1 1 ¯1)(1 ¯1 1 1)(1 1 ¯1 1)                         ⍝ product unit factors
    QP ← u{+/⍺⍺×[¯2↑⍳r+1]⍺×[(⍳(≢⍴⍺)-1),r+⎕IO](⊂⎕IO+⍵⍵)⌷[(r←≢⍴⍵)-1-⎕IO]⍵}c  ⍝ quaternion product
    QC ← ×⍤1∘(1,-3⍴1)                                                        ⍝ quaternion conjugate
    QD ← +.×⍤1∘QC                                                            ⍝ quaternion dot product

    ⍝ quaternion from Euler angles
    RD ← 180÷⍨○                                                              ⍝ radians from degrees
    UV ← {⍵×[⍳(≢⍴⍵)-1]÷(+/×⍨⍵)*÷2}                                           ⍝ unitary vector
    QA ← {(2○⍵),(1○⍵)∘.×UV⍺}∘(÷∘2) ⋄ QAD ← QA∘RD                             ⍝ quaternion from axis-angle
    QE ← {⍺←(0 0 1)(1 0 0)(0 0 1) ⋄ ⊃⍺QP.QA⊂[¯1↓⍳≢⍴⍵]⍵} ⋄ QED ← QE∘RD        ⍝ quaternion from Euler angles (zxz)

    ⍝ cubic symmetry
    cs ← ⊂1 0 0 0                                                            ⍝ identity
    cs,←,(1 0 0)(0 1 0)(0 0 1)                          ∘.QAD 90 180 270     ⍝ 4-fold around <001>
    cs,←,(1 1 1)(¯1 1 1)(1 ¯1 1)(1 1 ¯1)                ∘.QAD 120 240        ⍝ 3-fold around <111>
    cs,←,(1 1 0)(1 0 1)(0 1 1)(1 ¯1 0)(¯1 0 1)(0 1 ¯1)∘.QAD 180              ⍝ 2-fold around <110>

    ⍝ misorientations
    ML ← {⍺←0.5 ⋄ ⌊(180×⍵)÷○⍺}                                               ⍝ misorientation level from radians
    MC ← cs{2×¯2○1⌊⊃⌈⌿|⍺⍺∘.QD⊂⍺QP∘QC⍵}                                        ⍝ misorientation with cubic symmetry
    IM ← {⎕IO+1-⍨(⍺⌊⍵)++/1-⍨⍳(⍺⌈⍵)-1}                                        ⍝ index of misorientation
    CM ← {c←⎕NS⍬ ⋄ c.m←¯1⍴⍨(⊢IM-∘1)≢,↓⍵ ⋄ c}                                  ⍝ cache of misorientations
    _M_ ← {⍺=⍵:0 ⋄ 0≤m←(i←⍺IM⍵)⊃⍵⍵.m:m ⋄ ⊢(i⊃⍵⍵.m)←ML⍺(MC)⍤(⊃∘⍺⍺)⍵}          ⍝ memoization

    ⍝ namespace with function M to calculate misorientations
    M ← {⍺←0.5                                                               ⍝ misorientation step
        m←⎕NS'QD' 'QP' 'QC' 'IM' 'MC' ⋄ m.ML←⍺∘ML                             ⍝ namespace with curried ML
        m.M←(⍺÷2)+⍺×(,↓⍵)_M_(CM⍵) ⋄ 2≥≢⍴⍵: m                                 ⍝ misorientation function
        m.M←m.M⍤((¯1↓⍴⍵)∘{⎕IO+⍺⊥⍵-⎕IO}) ⋄ m                                  ⍝ over list index for higher rank
    }

    Space ← {↑⊃∘.,/(⍵÷2)+⍵×(⍳¨360 90 90÷⍵)-1}                                ⍝ Euler space of given step
    Random ← {F←{(1 2○2×⊂○?⍺⍴0)×⊂⍵*÷2} ⋄ UV↑[1]⍵(F∘(1∘-),F)?⍵⍴0}              ⍝ random orientations of given shape
:EndNamespace
```

Development

- Specialized tutorials: materials (jupyter)
    - Analysis of tensile experiments
    - Crystallographic orientations and misorientations
    - Grain growth

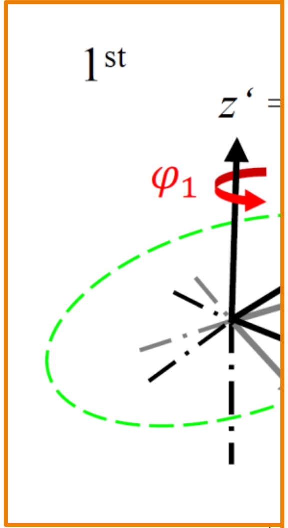
```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)                       ⍝ product components ⎕IO=0
    u ← ↑(1 ¯1 ¯1 ¯1)(1 1 1 ¯1)(1 ¯1 1 1)(1 1 ¯1 1)                  ⍝ product unit factors
    QP ← u{+/⍺⍺×[¯2↑⍳r+1]⍺×[(⍳(≢⍴⍺)-1),r+⎕IO](⊂⎕IO+⍵⍵)⌷[(r←≢⍴⍵)-1-⎕IO]⍵}c  ⍝ quaternion product
    QC ← ×⍤1∘(1,-3⍴1)                                                ⍝ quaternion conjugate
    QD ← +.×⍤1∘QC                                                    ⍝ quaternion dot product
                                                                      ngle
                                                                      angles (zxz)
                                                                      om radians
                                                                      ic symmetry
                                                                      n
                                                                      ns
                                                                      ML
                                                                      n
                                                                      gher rank
    }
    Space ← {↑⊃∘.,/(⍵÷2)+⍵×(⍳¨360 90 90÷⍵)-1}                        ⍝ Euler space of given step
    Random ← {F←{(1 2○2×⊂○?⍺⍴0)×⊂⍵*÷2} ⋄ UV↑[1]⍵(F∘(1∘-),F)?⍵⍴0}     ⍝ random orientations of given shape
:EndNamespace
```

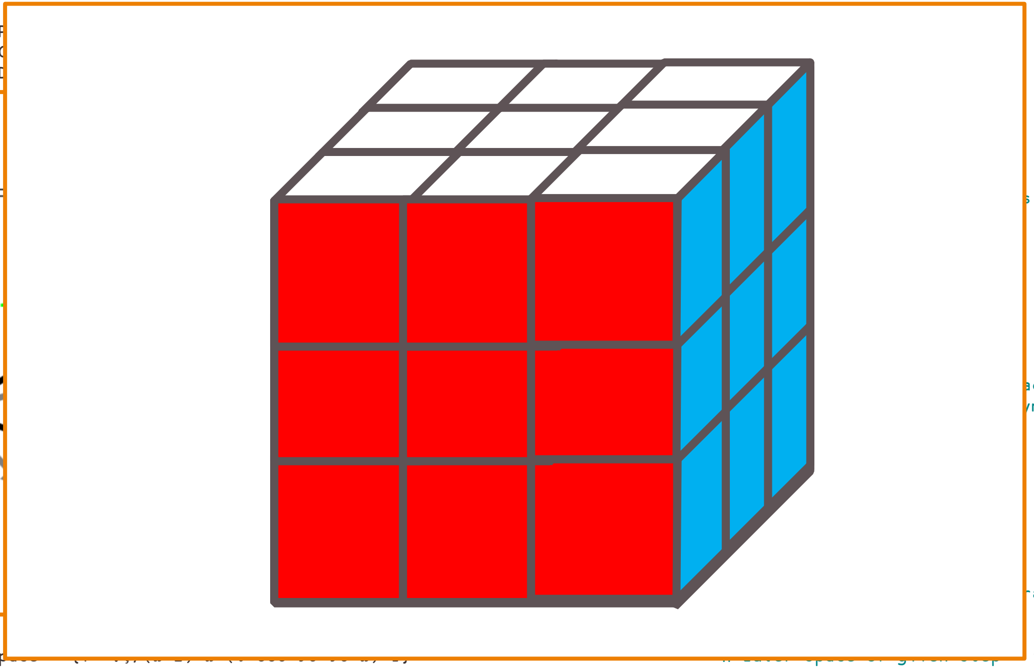


Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)          ⍝ product components ⎕IO=0
    u
    QF
    QC
    QD
    Sp
    Random ← {F←{(1 2∘2×⊂∘?α⍴0)×⊂ω*÷2} ⋄ UV↑[1]ω(F∘(1∘-),F)?ω⍴0}          ⍝ random orientations of given shape
:EndNamespace
```





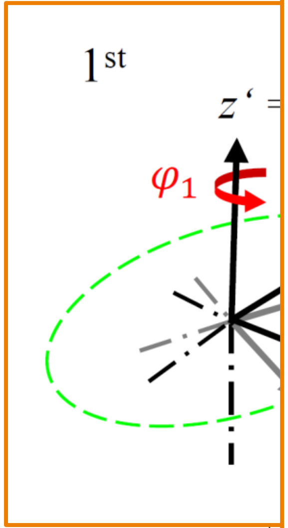
jgl@dyalog.com

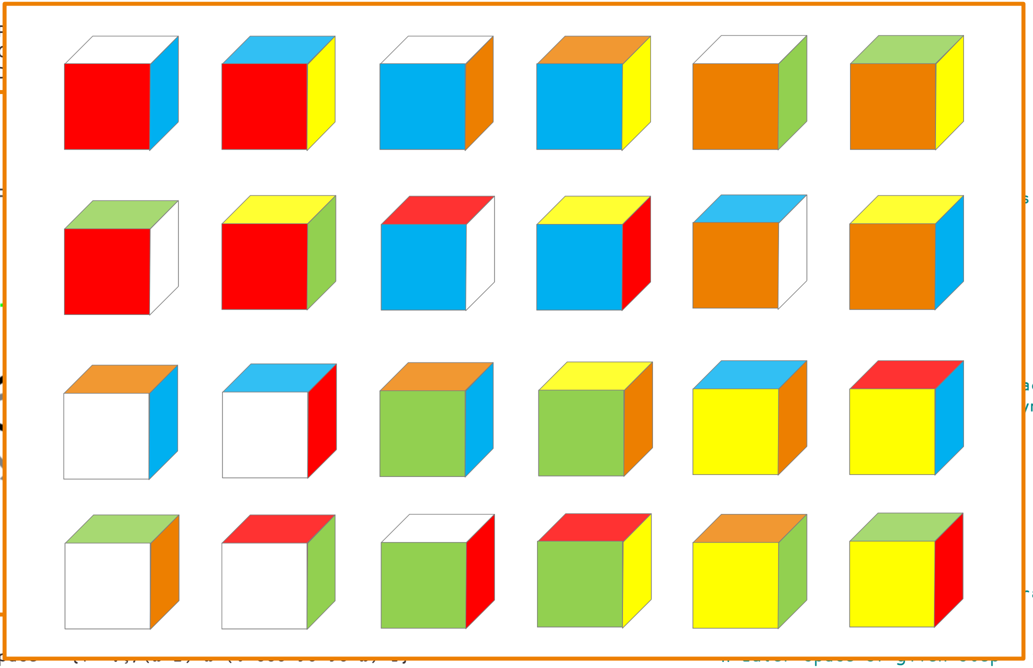
jesus.galanlopez@ugent.be

Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)                         ⍝ product components ⎕IO=0
    u
    QR
    QC
    QD
                                                                        s (zxz)
                                                                        adians
                                                                        ymmetry
                                                                        rank
    Sp
    Random ← {F←{(1 2○2×⊂○?αρ0)×⊂ω*÷2} ⋄ UV↑[1]ω(F∘(1∘-),F)?ωρ0}      ⍝ random orientations of given shape
:EndNamespace
```





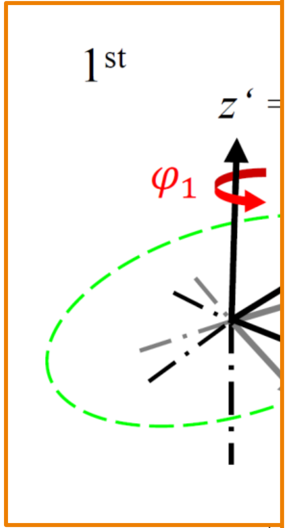
jgl@dyalog.com

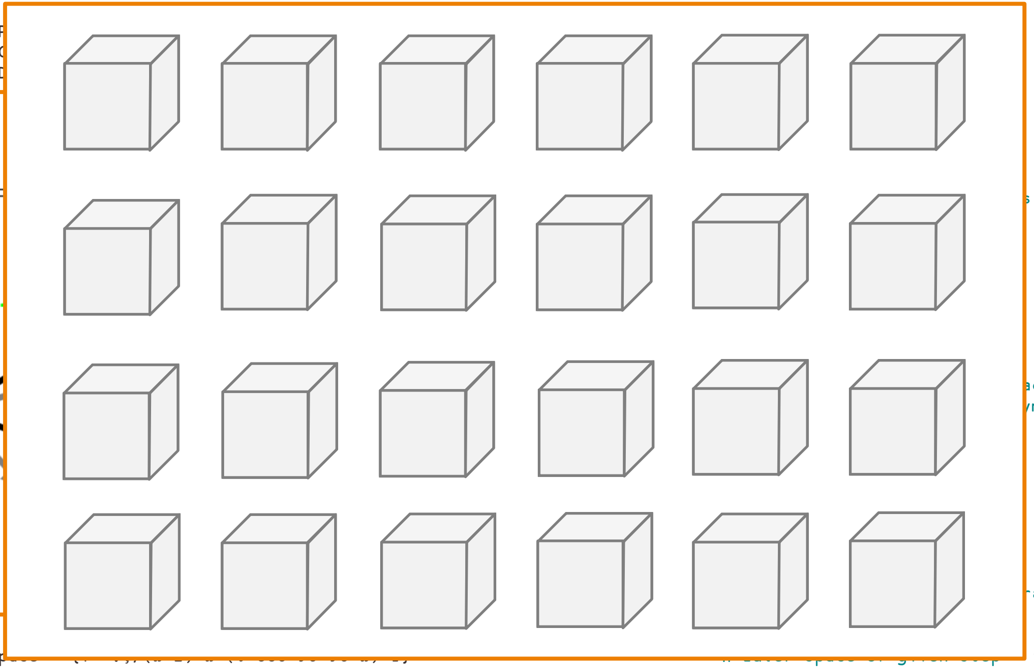
jesus.galanlopez@ugent.be

Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)                 ⍝ product components ⎕IO=0
    u
    QR
    QC
    QD
                                                                                     s (zxz)
                                                                                     adians
                                                                                     ymmetry
                                                                                     rank
    Sp
    Random ← {F←{(1 2○2×⊂○?αρ0)×⊂ω*÷2} ⋄ UV↑[1]ω(F∘(1∘-),F)?ωρ0}   ⍝ random orientations of given shape
:EndNamespace
```





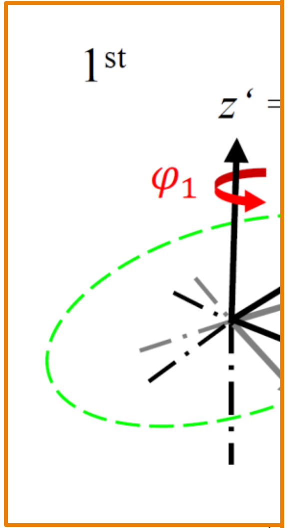
jgl@dyalog.com

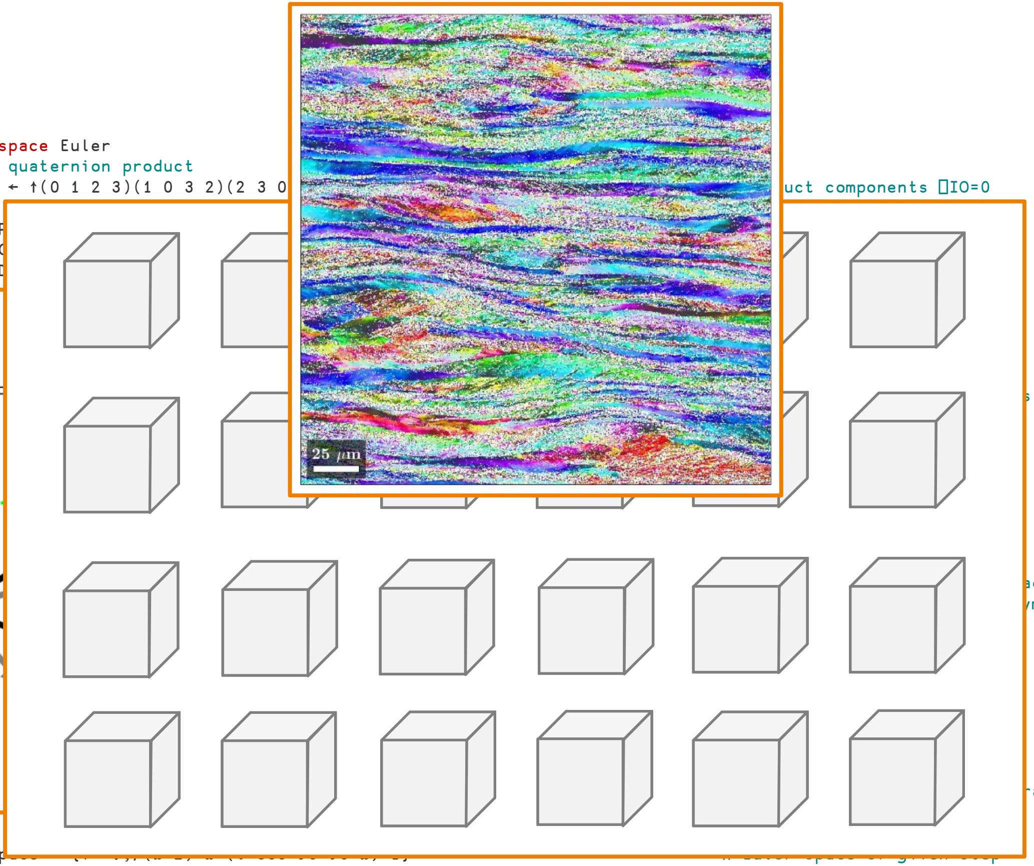
jesus.galanlopez@ugent.be

Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

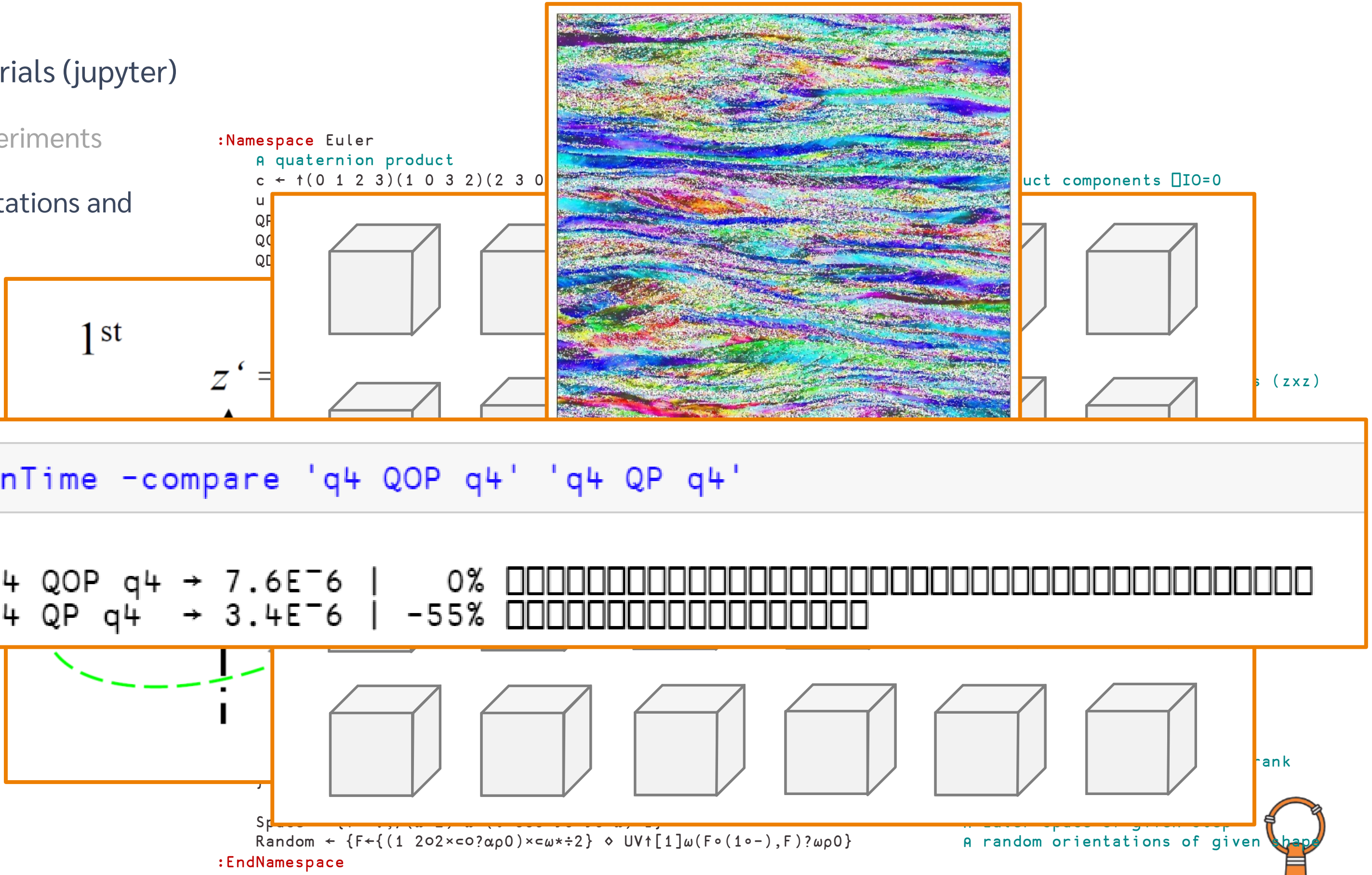
```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0                          uct components ⎕IO=0
    u
    QP
    QC
    QD
                                                                         s (zxz)
                                                                         adians
                                                                         mmetry
                                                                         ank
    Sp
    Random ← {F←{(1 2∘2×○?αρ0)×⊂ω×÷2} ⋄ UV↑[1]ω(F∘(1∘−),F)?ωρ0}    ⍝ random orientations of given shape
:EndNamespace
```





Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth



jgl@dyalog.com

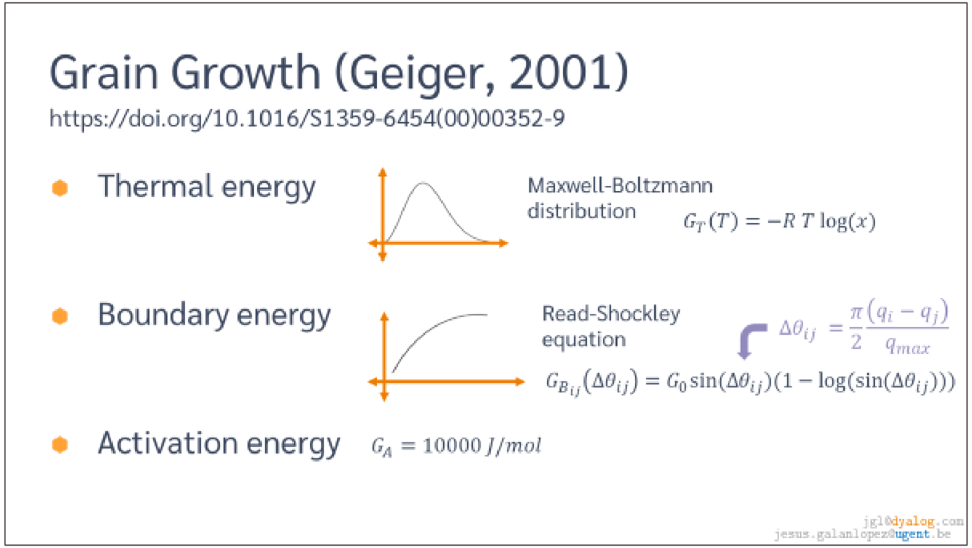
jesus.galanlopez@ugent.be

Development

- Specialized tutorials: materials (jupyter)
    - Analysis of tensile experiments
    - Crystallographic orientations and misorientations
    - Grain growth

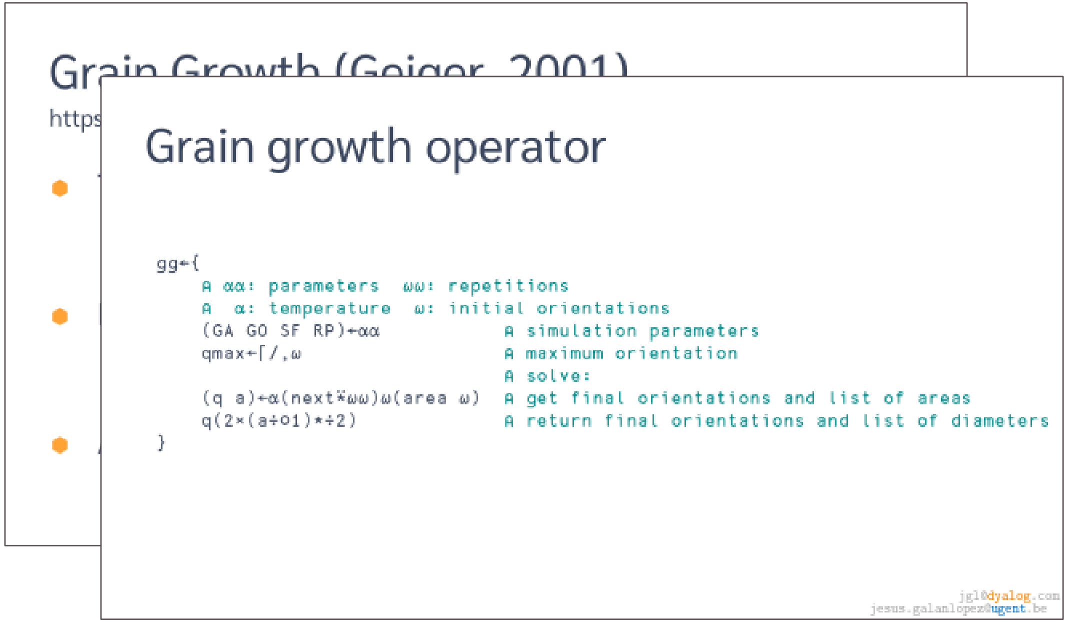
Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth



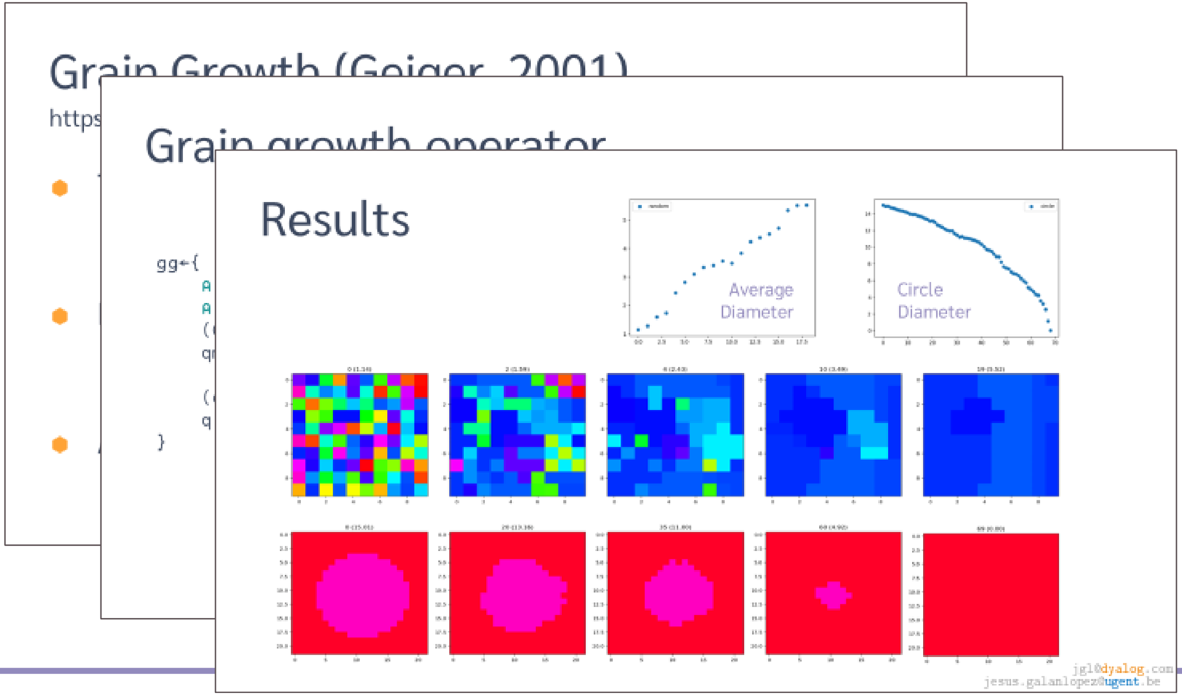
Development

- Specialized tutorials: materials (jupyter)
    - Analysis of tensile experiments
    - Crystallographic orientations and misorientations
    - Grain growth



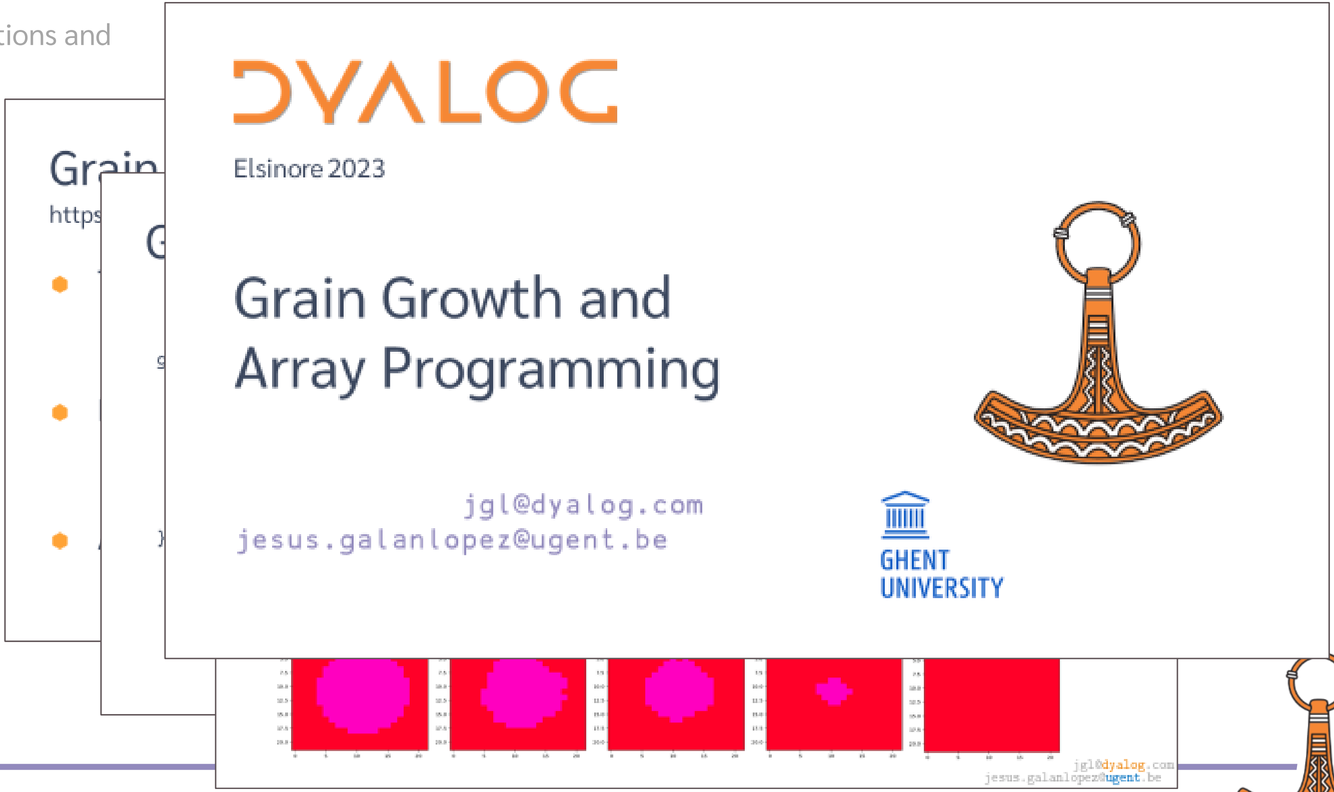
Development

- Specialized tutorials: materials (jupyter)
    - Analysis of tensile experiments
    - Crystallographic orientations and misorientations
    - Grain growth



Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth



Development

- Specialized tutorials: materials (jupyter)
  - Analysis of tensile experiments
  - Crystallographic orientations and misorientations
  - Grain growth

Research

- Geometric algebra
- Grain growth
- Crystal plasticity

Research

- Geometric algebra
- Grain growth
- Crystal plasticity

```
:Namespace ga ⍝                              Geometric Algebra
    ⎕IO ← 0                                                              ⍝ important
    ⎕CT ← 1e¯10                                                          ⍝ avoid rounding issues
    Assert ← {⍺←'assertion failure' ⋄ 0∊⍵:⍺ ⎕signal 8 ⋄ shy←1}           ⍝ Assert by Roger Hui

    ⍝ vectors
     M ← 0.5*⍨+.×⍨  ⋄   U ← ÷∘M⍨        ⋄   I ← ×∘(÷+.×⍨)⍨              ⍝ magnitude, unitary, inverse
    AV ← ⊂⍤1↑[-÷2]  ⋄  VA ← ⊂⍤¯1↑[-÷2]  ⋄  BV ← {1↑⍨¨-1+ι⍵}             ⍝ array from vector, vector from array, base

    ⍝ vector extension
     C ← {⍺←⎕CT ⋄ ⎕CT←⍺ ⋄ 1=≢∪⍵:⊃⍵ ⋄ ⍺∧.>|∊⍵:0 ⋄ ⍵}¨                   ⍝ collapse equal values
     S ← {θ≡⍵:0 ⋄ (1<≢⍵)∨(0<≡⍵)∧1<≢⊃⍵:⍵ ⋄ ⊃⍵}   ⋄ US ← ×∘(×/⍤=⍨)⍨         ⍝ scalar if possible and unscalar
     Z ← {⍺←⎕CT ⋄ ⎕CT←⍺ ⋄ S ⍵↓⍨-⊥⍨(1∧.=1+∊)¨⍵}  ⋄  E ← ≡⍤Z⍤C               ⍝ remove trailing zeros and equivalent
     X ← {⍺←2*D⍵ ⋄ ⍺↑⍵,0⍴⍨0⌈⍺-≢⍵}  ⋄  _X ← {⍺←⊢ ⋄ ⍺(⍺⍺⍤((⍺⌈⍤≢⍵)∘X))⍵}    ⍝ extend and apply extending arguments
    _V ← {⍺←⊢ ⋄ Z∘C ⍺(⍺⍺ _X)⍵}      ⋄  _Z ← {0≡⍺:0 ⋄ 0≡⍵:0 ⋄ ⍺(⍺⍺)⍵}      ⍝ apply vector extension or zero function

    ⍝ multivectors
     V ← {⍺←2*ι≢⍵ ⋄ r←⍺⌊⍤⍴⍵ ⋄ v←0⍴⍨1+⌈/,a←r↑⍺ ⋄ v[a]+←r↑⍵ ⋄ v}          ⍝ multivector
    BB ← ↓∘⍉2⊥⍣¯1ι  ⋄  G ← {⌈/+/¨(0≢¨C ⍵)/BB≢⍵}  ⋄  D ← ⌈2⍟⌈⍤≢           ⍝ binary unitary base, grade and dimensions
    GB ← {b←Z({⊂⊂⍵}⌸+/¨BB≢⍵)⌷¨⊂,⍵ ⋄ ⍺←ι≢b ⋄ (⊂⍺)⌷(1+⌈/,⍺)X b}          ⍝ group in blades

    ⍝ product tables
     PB ← 2⊥≢  ⋄  SB ← ¯1*(≠\¯1↓⊢)≠.∧1↓⊣  ⋄  MS ← ⊃1 ¯1 0,.⍴⍨⊢↑⍨3⌈≢      ⍝ position, sign and metric
    _FM ← {(⍺SB⍵)×(⌽⍺⍺)×.*⍺∧⍵}  ⋄  TS ← {(∘.PB,⍤⊂∘.((MS⍵)_FM))⍨BB 2*+/⍵}  ⍝ factor from metric, tables from signature
     TX ← TS 0 0,+/  ⋄  TI ← 0 0⍉∘⊃⌽∘TS  ⋄  TR ← SB⍨¨BB∘≢  ⋄  BT ← ⊃V¨/   ⍝ exterior, inner, reversion and base

    ⍝ geometric operators
    _Δ_  ← {p s←⍺(⍵⍵⍨⍨)⍵ ⋄ (r↑p)V(r↑s)×(r←(≢¨⍺⍵)⌊⍴p)↑⍺∘.(⍺⍺_Z)⍤,⍵}    ⍝ geometric product
    _ΔI_ ← {0 V _X ⍺(⍵⍵×_X⍺⍺_Z¨_X)⍵}      ⋄  _ΔX_ ← {Z⍺(⍺⍺_Δ_⍵⍵)⍵}         ⍝ inner and exterior product
    _Δ   ← _Δ_(TS⍤D)  ⋄  _ΔX ← _Δ_(TX⍤D)  ⋄  _ΔI ← _ΔI_(TI⍤D)             ⍝ geometric, inner and exterior product
    _⍙_  ← {⍵⍵ LC ⍺ ⍺⍺⍤((⍺(⍵⍵⍨⍨)⍵)∘RC)⍵}  ⋄  _⍙ ← _⍙_ D                   ⍝ antigeometric product
    _I   ← {⍺←1 ⋄ Z∘C⍺ ⍺⍺ ⊂∘⍉⍤¯1⍉(↑∘1∘≢⌸⍤2(↑[-÷2]⍣2⊂⍺⍺¨BV∘≢))US⍵X⍨2*D⍵}  ⍝ inverse
    _C   ← {⍺←D⍵ ⋄ (⌽×_V+/⍤⍺⍺¨∘⌽⍨∘BV∘≢)⍵ X⍨2*⍺}                          ⍝ complement

    ⍝ geometric functions
    Δ  ← ×_Δ    ⋄  ΔX ← ×_ΔX   ⋄  ΔI ← ×_ΔI                               ⍝ geometric, exterior and inner product
    ⍙  ← Δ _⍙   ⋄  ⍙X ← ΔX _⍙  ⋄  ⍙I ← ΔI _⍙                              ⍝ antigeometric, exterior and inner product
    LC ← Δ _C   ⋄  RC ← Δ⍨ _C  ⋄  DC ← LC⍣2                               ⍝ left, right and double complement
    ΔΔ ← Δ _I   ⋄  ⍙⍙ ← ⍙ _I   ⋄  R  ← ×∘TR⍨  ⋄  PS ← 1↑⍨∘-2*∘⌈2⍟≢       ⍝ inverse, antiinverse, reverse, pseudoscalar

    GA ← {                                                                ⍝ GEOMETRIC ALGEBRA
        g ← ⎕NSθ  ⋄  g.s g.b ← ⍵ (S BV 2*d←+/⍵)                           ⍝   namespace, signature, base
        g._Z←_Z ⋄ g._X←_X ⋄ g.Z←Z ⋄ g.S←S ⋄ g.US←US ⋄ g.C←C
        g.V←V ⋄ g._V←_V ⋄ g.D←D ⋄ g.BV←BV ⋄ g._⍙←_⍙ d
        g._Δ ← _Δ_(TS⍵)  ⋄  g._ΔX ← _Δ_(TX⍵)  ⋄  g._ΔI ← _ΔI_(TI⍵)        ⍝   geometric, exterior and inner operator
        g.Δ  ← ×g._Δ     ⋄  g.ΔX  ← ×g._ΔX    ⋄  g.ΔI  ← ×g._ΔI           ⍝   products
        g.ΔV ← g.Δ⍤V     ⋄  g.ΔXV ← g.ΔX⍤V    ⋄  g.ΔIV ← g.ΔI⍤V           ⍝   products of vectors
        g.Δv ← g.Δ∘V     ⋄  g.ΔXv ← g.ΔX∘V    ⋄  g.ΔIv ← g.ΔI∘V           ⍝   products by vector
        g.⍙  ← g.Δ g._⍙  ⋄  g.⍙X  ← g.ΔX g._⍙ ⋄  g.⍙I  ← g.ΔI g._⍙        ⍝   antiproducts
        g.ΔΔ ← g.Δ _I    ⋄  g.ΔΔV ← g.ΔΔ⍤V    ⋄  g.ΔΔv ← g.ΔΔ∘V           ⍝   inverse and division
        g.LC ← g.Δ _C    ⋄  g.RC  ← g.Δ⍨_C    ⋄  g.DC  ← g.LC⍣2           ⍝   complements: left, right, double
        g.R  ← ×∘(TR⍵)⍨  ⋄  g.X   ← X{⍺←≢b ⋄ ⍺(⍺⍺)⍵}  ⋄  g                ⍝   reverse and extend
    }

    g0 g1 g2 g3 c q pga ← GA¨(ι4),(0 1) (0 2) (3 0 1)                     ⍝ 0-3D, complex, quaternions, pga

    VS ← {                                                                ⍝ Vector Space
        _ ← Assert (⊂θ) ∧.≡ ⍴¨ a b ← ⍺                                    ⍝   a and b are scalars
        _ ← Assert (⊂,1) ∧.≥ ⍴∘⍴¨ u v w ← ⍵                               ⍝   u, v and w are vectors (rank 1)
        _ ← Assert v                  E v + 0                             ⍝   identity element for addition
        _ ← Assert v                  E 1 × v                             ⍝   identity element for multiplication
        _ ← Assert 0                  E v +_V -v                          ⍝   inverse element for addition
        _ ← Assert (v +_V w)          E w +_V v                           ⍝   commutative addition
        _ ← Assert (a × v)            E v × a                             ⍝   commutative product with scalars
        _ ← Assert (u +_V v +_V w) E w +_V u +_V v                        ⍝   associative addition
        _ ← Assert (v × a × b)        E (v × a) × b                       ⍝   associative product with scalars
        _ ← Assert (a × v +_V w)      E v +_V⍤(a∘×) w                     ⍝   distributive over addition of vectors
            Assert (v × a + b)        E a +_V⍤(v∘×) b                     ⍝   distributive over addition of scalars
    }
    _GA ← {                                                               ⍝ Geometric Algebra (geometric product ⍺⍺)
        _ ← Assert (a b ← ⍺)           VS (u v w ← ⍵)                     ⍝   vector space: a b scalars, u v w vectors
        _ ← Assert (u ⍺⍺ v ⍺⍺ w)       E w ⍺⍺⍨ u ⍺⍺ v                     ⍝   associative product
        _ ← Assert (u∘⍺⍺ ⊃+_V/ v w) E (⊃+_V/ u∘⍺⍺¨ v w)                  ⍝   distributive by left
        _ ← Assert (⍺⍺∘u ⊃+_V/ v w) E (⊃+_V/ ⍺⍺∘u¨ v w)                  ⍝   distributive by right
        _ ← Assert (a ⍺⍺ v ⍺⍺ b)       E a × b × v                        ⍝   extension of scalar product
            Assert (⊢Eθ⍴⊢)∘(⍺⍺⍨)¨      V¨BV⌈/D¨⍵                          ⍝   scalar product of parallel vectors
    }
:EndNamespace                                                             ⍝ jgl@dyalog.com 2022
```

- Geometric algebra
- Grain growth
- Crystal plasticity

```
GA ← {                                                                            ⍝ GEOMETRIC ALGEBRA
    g ← ⎕NS⍬  ⋄  g.s g.b ← ω (S BV 2*d←+/ω)                                       ⍝   namespace, signature, base
    g._Z←_Z ⋄ g._X←_X ⋄ g.Z←Z ⋄ g.S←S ⋄ g.US←US ⋄ g.C←C
    g.V←V ⋄ g._V←_V ⋄ g.D←D ⋄ g.BV←BV ⋄ g._⍙←_⍙ d
    g._Δ ← _Δ_(TSω)   ⋄  g._ΔX ← _Δ_(TXω)   ⋄  g._ΔI ← _ΔI_(TIω)                 ⍝   geometric, exterior and inner operator
    g.Δ  ← ×g._Δ      ⋄  g.ΔX  ← ×g._ΔX     ⋄  g.ΔI  ← ×g._ΔI                    ⍝   products
    g.ΔV ← g.Δ⍥V      ⋄  g.ΔXV ← g.ΔX⍥V     ⋄  g.ΔIV ← g.ΔI⍥V                    ⍝   products of vectors
    g.Δv ← g.Δ∘V      ⋄  g.ΔXv ← g.ΔX∘V     ⋄  g.ΔIv ← g.ΔI∘V                    ⍝   products by vector
    g.⍙  ← g.Δ g._⍙   ⋄  g.⍙X  ← g.ΔX g._⍙  ⋄  g.⍙I  ← g.ΔI g._⍙                 ⍝   antiproducts
    g.ΔΔ ← g.Δ _I     ⋄  g.ΔΔV ← g.ΔΔ⍥V     ⋄  g.ΔΔv ← g.ΔΔ∘V                    ⍝   inverse and division
    g.LC ← g.Δ _C     ⋄  g.RC  ← g.Δ⍨_C     ⋄  g.DC  ← g.LC⍣2                    ⍝   complements: left, right, double
    g.R  ← ×∘(TRω)⍨   ⋄  g.X   ← X{α←≢b ⋄ α(αα)ω}  ⋄  g                          ⍝   reverse and extend
}
```


jgl@dyalog.com

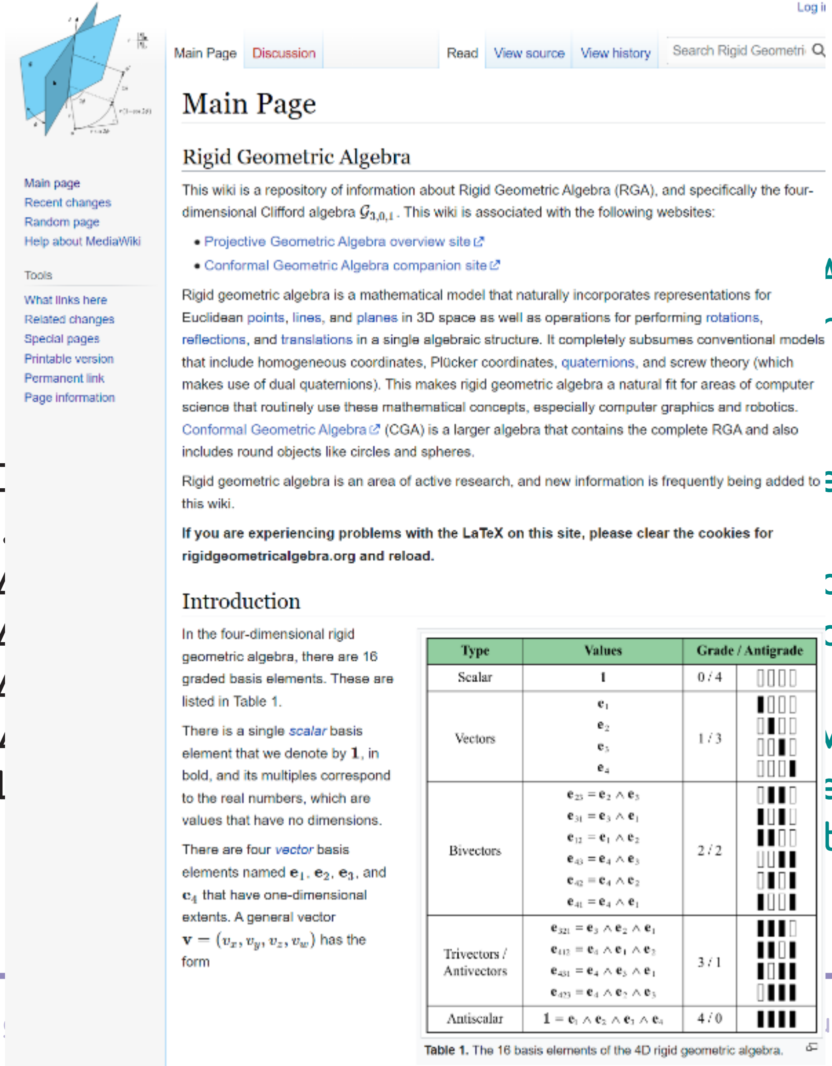
jesus.galanlopez@ugent.be

- Geometric algebra
- Grain growth
- Crystal plasticity

```
                                   ⍝ GEOMETRIC ALGEBRA
ω)                                 ⍝   namespace, signature, base
.US←US ⋄ g.C←C
._⍙←_⍙_ d

  ⋄   g._∆I ← _∆I_(TIω)            ⍝   geometric, exterior and inner operator
  ⋄   g.∆I  ← ×g._∆I               ⍝   products
  ⋄   g.∆IV ← g.∆I⍤V               ⍝   products of vectors
  ⋄   g.∆Iv ← g.∆I∘V               ⍝   products by vector
  ⋄   g.⍙I  ← g.∆I g._⍙            ⍝   antiproducts
  ⋄   g.∆∆v ← g.∆∆∘V               ⍝   inverse and division
  ⋄   g.DC  ← g.LC⍣2               ⍝   complements: left, right, double
α(αα)ω}  ⋄  g                      ⍝   reverse and extend
```

Research

- Geometric algebra
- Grain growth
- Crystal plasticity



Log in

Main Page | Discussion | Read | View source | View history | Search Rigid Geometri

Main page
Recent changes
Random page
Help about MediaWiki

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information

Main Page

Rigid Geometric Algebra

This wiki is a repository of information about Rigid Geometric Algebra (RGA), and specifically the four-dimensional Clifford algebra $\mathcal{G}_{3,0,1}$. This wiki is associated with the following websites:

- Projective Geometric Algebra overview site
- Conformal Geometric Algebra companion site

Rigid geometric algebra is a mathematical model that naturally incorporates representations for Euclidean points, lines, and planes in 3D space as well as operations for performing rotations, reflections, and translations in a single algebraic structure. It completely subsumes conventional models that include homogeneous coordinates, Plücker coordinates, quaternions, and screw theory (which makes use of dual quaternions). This makes rigid geometric algebra a natural fit for areas of computer science that routinely use these mathematical concepts, especially computer graphics and robotics. Conformal Geometric Algebra (CGA) is a larger algebra that contains the complete RGA and also includes round objects like circles and spheres.

Rigid geometric algebra is an area of active research, and new information is frequently being added to this wiki.

If you are experiencing problems with the LaTeX on this site, please clear the cookies for rigidgeometricalgebra.org and reload.

Introduction

In the four-dimensional rigid geometric algebra, there are 16 graded basis elements. These are listed in Table 1.

There is a single *scalar* basis element that we denote by **1**, in bold, and its multiples correspond to the real numbers, which are values that have no dimensions.

There are four *vector* basis elements named $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, and $\mathbf{e}_4$ that have one-dimensional extents. A general vector $\mathbf{v} = (v_x, v_y, v_z, v_w)$ has the form

Type	Values	Grade / Antigrade
Scalar	$\mathbf{1}$	0 / 4
Vectors	$\mathbf{e}_1$ $\mathbf{e}_2$ $\mathbf{e}_3$ $\mathbf{e}_4$	1 / 3
Bivectors	$\mathbf{e}_{23} = \mathbf{e}_2 \wedge \mathbf{e}_3$ $\mathbf{e}_{31} = \mathbf{e}_3 \wedge \mathbf{e}_1$ $\mathbf{e}_{12} = \mathbf{e}_1 \wedge \mathbf{e}_2$ $\mathbf{e}_{43} = \mathbf{e}_4 \wedge \mathbf{e}_3$ $\mathbf{e}_{42} = \mathbf{e}_4 \wedge \mathbf{e}_2$ $\mathbf{e}_{41} = \mathbf{e}_4 \wedge \mathbf{e}_1$	2 / 2
Trivectors / Antivectors	$\mathbf{e}_{321} = \mathbf{e}_3 \wedge \mathbf{e}_2 \wedge \mathbf{e}_1$ $\mathbf{e}_{412} = \mathbf{e}_4 \wedge \mathbf{e}_1 \wedge \mathbf{e}_2$ $\mathbf{e}_{431} = \mathbf{e}_4 \wedge \mathbf{e}_3 \wedge \mathbf{e}_1$ $\mathbf{e}_{423} = \mathbf{e}_4 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$	3 / 1
Antiscalar	$\bar{\mathbf{1}} = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 \wedge \mathbf{e}_4$	4 / 0

Table 1. The 16 basis elements of the 4D rigid geometric algebra.

https://rigidgeometricalgebra.org/wiki/

ω)
.US←US ⋄ g.C←C
._Δ←_Δ_ d
⋄ g._ΔI ← _Δ
⋄ g.ΔI ← ×g
⋄ g.ΔIV ← g.
⋄ g.ΔIv ← g.
⋄ g.ΔI ← g.
⋄ g.ΔΔv ← g.
⋄ g.DC ← g.
α(αα)ω} ⋄ g

A
nature, base
erior and inner operator
ctors
ctor
vision
eft, right, double
tend

j... us.galanlopez@ugent.be

- Geometric algebra
- Grain growth
- Crystal plasticity

```
⍝ geometric product operator
_GP_ ← {t ← α(ωω⍨⍨)ω ⋄ +⌿↑,α(t×∘⊂¨(ρt)↑∘.(αα_Z))ω}
_GP  ← _GP_(BN⌈⍤≢)
```

v1	v2	×_GP
1 2	3 4	11 0 0 2

- Geometric algebra
- Grain growth
- Crystal plasticity

```
⍝ geometric product operator
_GP_ ← {t ← ⍺(⍵⍵⍨⍨)⍵ ⋄ +⌿↑,⍺(t×∘⊂¨(⍴t)↑∘.(⍺⍺_Z))⍵}
_GP  ← _GP_(BN⌈ö≢)
```

```
┌─────────┬─────────┬───────────────────┐
│v1       │v2       │×_GP               │
├─────────┼─────────┼───────────────────┤
│┌───┬───┐│┌───┬───┐│┌─────┬───┬───┬───┐│
││1 2│3 4│││5 6│7 8│││26 44│0 0│0 0│8 8││
│└───┴───┘│└───┴───┘│└─────┴───┴───┴───┘│
└─────────┴─────────┴───────────────────┘
```

- Geometric algebra
- Grain growth
- Crystal plasticity

```
⍝ geometric product operator
_GP_ ← {t ← ⍺(⍵⍵⍨⍨)⍵ ⋄ +⌿↑,⍺(t×∘⊂¨(⍴t)↑∘.(⍺⍺_Z))⍵}
_GP  ← _GP_(BN⌈⍥≠)
```

v1	v2	×_GP	∘.×_GP	+.×_GP
1 2 | 3 4	5 6 | 7 8	26 44 | 0 0 | 0 0 | 8 8	26 30 | 0 0 | 0 0 | 8 10 38 44 | 0 0 | 0 0 | 6 8	70 0 0 16

- Geometric algebra
- Grain growth
- Crystal plasticity

```
⍝ geometric product operator
_GP_ ← {t ← ⍺(⍵⍵⍨⍨)⍵ ⋄ +⌿↑,⍺(t×∘⊂¨(⍴t)↑∘.(⍺⍺_Z))⍵}
_GP  ← _GP_(BN⌈⍤≢)
```

z1	z2	×_GP	∘.×_GP	+.×_GP
[1 2] [3 4]	[5 6] [7 8]	[¯16 ¯20] [22 40]	[¯16 ¯18 / ¯18 ¯20] [22 26 / 34 40]	¯36 62

Research

- Geometric algebra
- Grain growth
- Crystal plasticity

z1	z2	×	∘.×	+.×
1J3 2J4	5J7 6J8	¯16J22 ¯20J40	¯16J22 ¯18J26 ¯18J34 ¯20J40	¯36J62

<table>
<tr><th>z1</th><th>z2</th><th>×_GP</th><th>∘.×_GP</th><th>+.×_GP</th></tr>
<tr>
<td><table><tr><td>1 2</td><td>3 4</td></tr></table></td>
<td><table><tr><td>5 6</td><td>7 8</td></tr></table></td>
<td><table><tr><td>¯16 ¯20</td><td>22 40</td></tr></table></td>
<td><table><tr><td>¯16 ¯18
¯18 ¯20</td><td>22 26
34 40</td></tr></table></td>
<td>¯36 62</td>
</tr>
</table>

Research

- Geometric algebra
- Grain growth
- Crystal plasticity

```
GrainGrowth
```

- Geometric algebra
- Grain growth
- Crystal plasticity

GrainGrowth

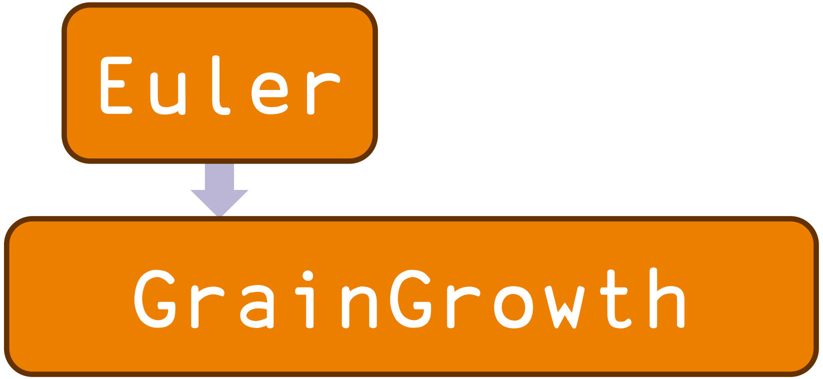
```
:Namespace GrainGrowth
    R←8.314 ⋄ Area←#.EBSD.Area
    _B←{⊃+⌿G0×s×1-⍟@(0∘<)¨s←1○180÷⍨○(⊢×GB<⊢)⊃⍺⍺/
    B1←{⍺(¯1⊖⍺)⍵(¯1⌽⍵)}_B
    B2←{⍺(¯1⊖¯1⌽⍺)⍵(¯1⊖1⌽⍵)}_B
    Step←{
        gt←-R×⍺×⍟?≠⍨⍵ ⋄ q1←1 ¯1(⊖¨,⌽¨)⊂⍵ ⋄ gb←⍵
        v1←1 ¯1(⊖¨,⌽¨)⊂v ⋄ t←(⊃∨/v1)∧(⊃(⊂⍵)∨.≠q
        gb1←(1+RP×?≠⍨q1)×{⍵+SF×q1 B2¨{(1⌽⍺)(¯1⊖
        (v1 gb1 q1)←{t/⍤,⍉↓⍉↑⍵}¨v1 gb1 q1 ⋄ m←⊃
    }
    Next←{q Step⍨←⍵ ⋄ a,←Area q ⋄ q}

    FromEBSD←{
        ⍺←10000 3000 5 0.1 0.1 1000 0.25 0.01 ⋄
        gg.g gg.q←gg.DA #.EBSD.Orientations ⍵ ⋄
        gg.v←(⍴gg.q)⍴(gg.IQ<#.EBSD.IQ ⍵)∧gg.CI<#
    }
    ToEBSD←{ebsd←⍵ ⋄ ebsd[⍳3]←↓⍉↑g[,q] ⋄ ebsd}

    Solve←{a⊣Next⍣≡⍵}
    _Solve_←{
         ⍺←1 ⋄ f←⍺ ⋄ n←0 ⋄ ebsd←⍺⍺ ⋄ name←⍵⍵
         a⊣Next⍣{n+←1 ⋄ (⍺≡⍵)⊣(ToEBSD ebsd)#.EB
    }
:EndNamespace
```


jgl@dyalog.com

jesus.galanlopez@ugent.be

Research

- Geometric algebra
- Grain growth
- Crystal plasticity



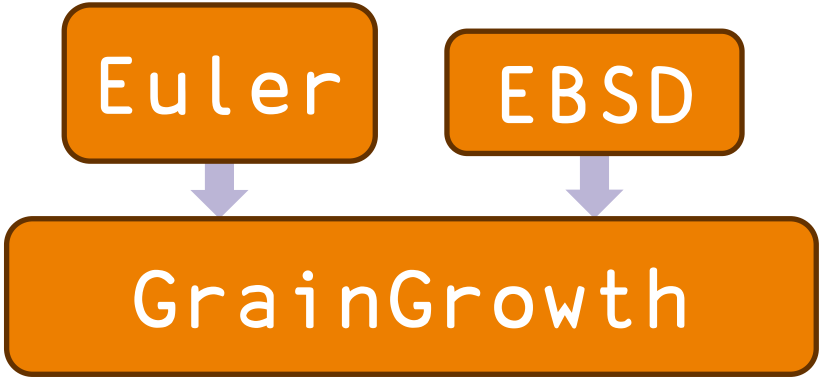

```
:Namespace Euler
    ⍝ quaternion product
    c ← ↑(0 1 2 3)(1 0 3 2)(2 3 0 1)(3 2 1 0)
    u ← ↑(1 ¯1 ¯1 ¯1)(1 1 1 ¯1)(1 ¯1 1 1)(1 1 ¯1 1)
    QP ← u{+/⍺⍺×[¯2↑⍳r+1]⍺×[(⍳(≢⍴⍺)-1),r+⎕IO](⊂⎕IO+⍵⍵)⌷[(r←≢⍴⍵)-1
    QC ← ×⍤1∘(1,-3⍴1)
    QD ← +.×⍤1∘QC

    ⍝ quaternion from Euler angles
    RD ← 180÷⍨○
    UV ← {⍵×[⍳(≢⍴⍵)-1]÷(+/×⍨⍵)*÷2}
    QA ← {(2○⍵),(1○⍵)∘.×UV⍺}∘(÷∘2) ⋄ QAD ← QA∘RD
    QE ← {⍺←(0 0 1)(1 0 0)(0 0 1) ⋄ ⊃⍺QP.QA⊂[¯1↓⍳≢⍴⍵]⍵} ⋄ QED ← Q

    ⍝ cubic symmetry
    cs ← ⊂1 0 0 0
    cs,←,(1 0 0)(0 1 0)(0 0 1)                          ∘.QAD 90 18
    cs,←,(1 1 1)(¯1 1 1)(1 ¯1 1)(1 1 ¯1)                ∘.QAD 120 2
    cs,←,(1 1 0)(1 0 1)(0 1 1)(1 ¯1 0)(¯1 0 1)(0 1 ¯1)∘.QAD 180

    ⍝ misorientations
    ML ← {⍺←0.5 ⋄ ⌊(180×⍵)÷○⍺}
    MC ← cs{2×¯2○1⌊⊃⌈⌿|⍺⍺∘.QD⊂⍺QP∘QC⍵}
    IM ← {⎕IO+1-⍨(⍺⌊⍵)++/1-⍨⍳(⍺⌈⍵)-1}
    CM ← {c←⎕NS⍬ ⋄ c.m←¯1⍴⍨(⊢IM-∘1)≢,↓⍵ ⋄ c}
    _M_ ← {⍺=⍵:0 ⋄ 0≤m←(i←⍺IM⍵)⊃⍵⍵.m:m ⋄ ⊢(i⊃⍵⍵.m)←ML⍺(MC)⍥(⊃∘⍺⍺)

    ⍝ namespace with function M to calculate misorientations
    M ← {⍺←0.5
        m←⎕NS'QD' 'QP' 'QC' 'IM' 'MC' ⋄ m.ML←⍺∘ML
        m.M←(⍺÷2)+⍺×(,↓⍵)_M_(CM⍵) ⋄ 2≥≢⍴⍵: m
        m.M←m.M⍥((¯1↓⍴⍵)∘{⎕IO+⍺⊥⍵-⎕IO}) ⋄ m
    }

    Space ← {↑⊃∘.,/(⍵÷2)+⍵×(⍳¨360 90 90÷⍵)-1}
    Random ← {F←{(1 2∘2×⊂○?⍺⍴0)×⊂⍵*÷2} ⋄ UV↑[1]⍵(F∘(1∘÷),F)?⍵⍴0}
:EndNamespace
```


jgl@dyalog.com

jesus.galanlopez@ugent.be

Research

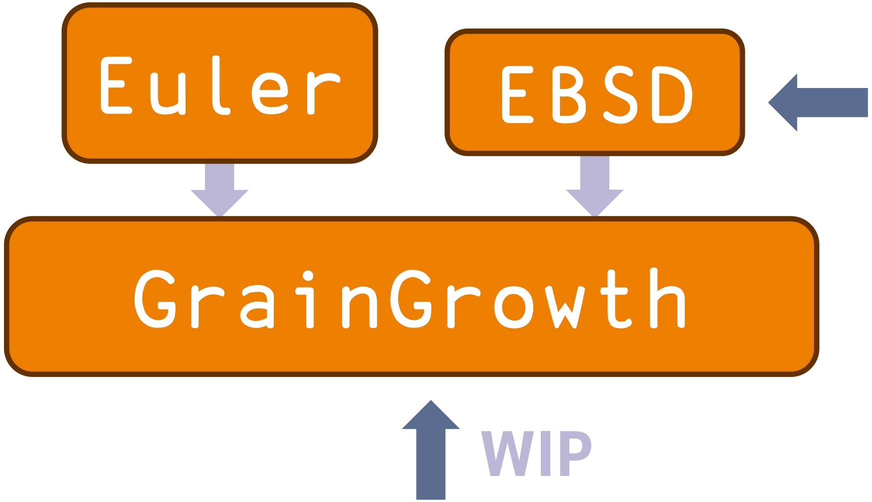
- Geometric algebra
- Grain growth
- Crystal plasticity




```
:Namespace EBSD
    Read←{  ⍝ read whitespace separated data f
        f←80 ¯1 ⎕MAP ⍵                         ⍝ map
        lf cr←⎕UCS 10 13                        ⍝ line
        clean←,/⊂⍤̃∘(≠∘'#'∧1,¯1↓=∘lf)⍤̃   ⍝ spli
        s←'\s+'⎕R','⊢'(^\s+)|(\s+$)'⎕R''⊢⊃clea
        ⎕CSV⍠'Invert' 2⊢s'S' 2 0
    }

    Write←{('#',¨⍺)⎕CSV⍠('Invert' 2)('Separato

    Crop←{x y←⍺ ⋄ s←((x>5∘⊃)∧y>4∘⊃)⍵ ⋄ s∘/¨⍵}

    Orientations←{
        x←5⊃⍵ ⋄ nx←⌊0.5+1+(⌈/x)÷⌈/|2-/x ⋄ ny←⌊
        ⍺←0 ⋄ ea←↓⍉↑3↑⍵ ⋄ ea←⍺(⌊0.5+÷⍨)⍣(⍺>0)⊢
    }

    IQ←6∘⊃  ⍝ image quality
    CI←7∘⊃  ⍝ confidence index

    Area←{(⍳2)×.((≢,⍵){⍺⍺÷1⌈+/,⍵≠1⌽[⍺]⍵})⊂⍵}
:EndNamespace
```

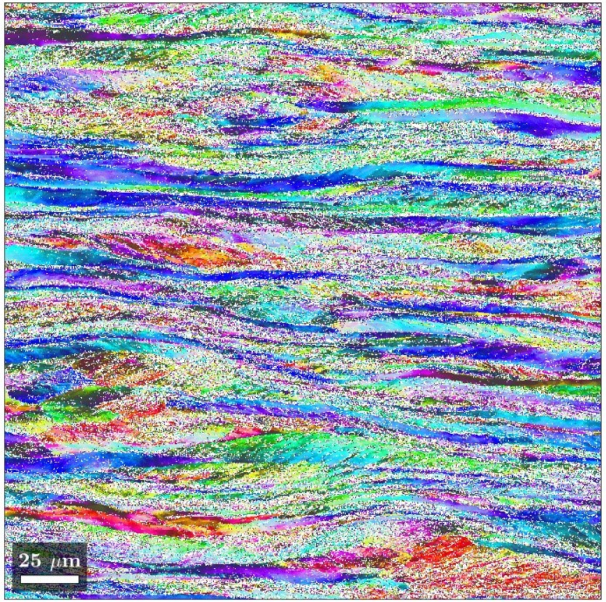
Research

- Geometric algebra
- Grain growth
- Crystal plasticity



UNIVERSITEIT GENT

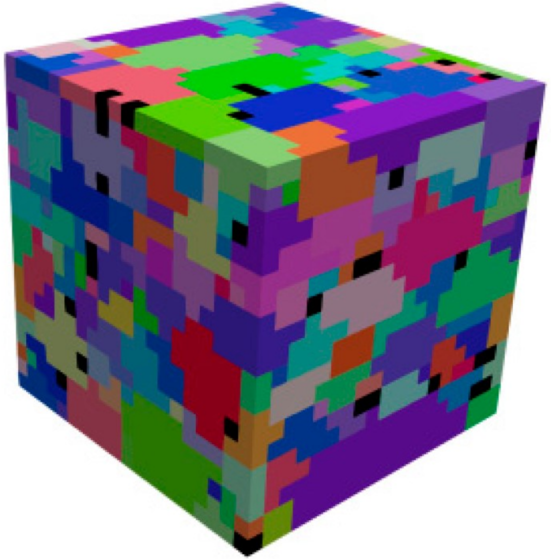
$$r_{0i} = \frac{M_{0i}}{V_0^2}\sqrt{\sum_d \left(\sum_j \delta_{ij} A_{0j} |\bar{u}_j \cdot \bar{u}_d|\right)^2} \sum_j (\gamma_{0j} - (1-\delta_{ij}) c_{0ij}\gamma_{ij}) A_{0j}$$



Sample preparation, microscopy: Estefanía Sepulveda
MTEX processing: JGL

Research

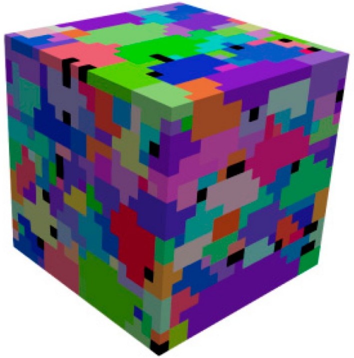
- Geometric algebra
- Grain growth
- Crystal plasticity

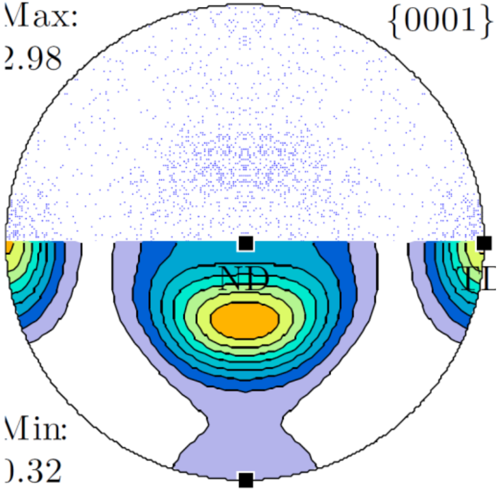


https://doi.org/10.3390/cryst10090819

Research

- Geometric algebra
- Grain growth
- Crystal plasticity

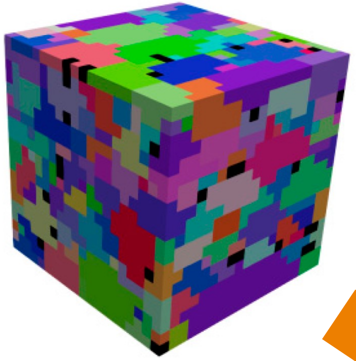


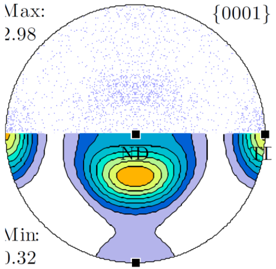


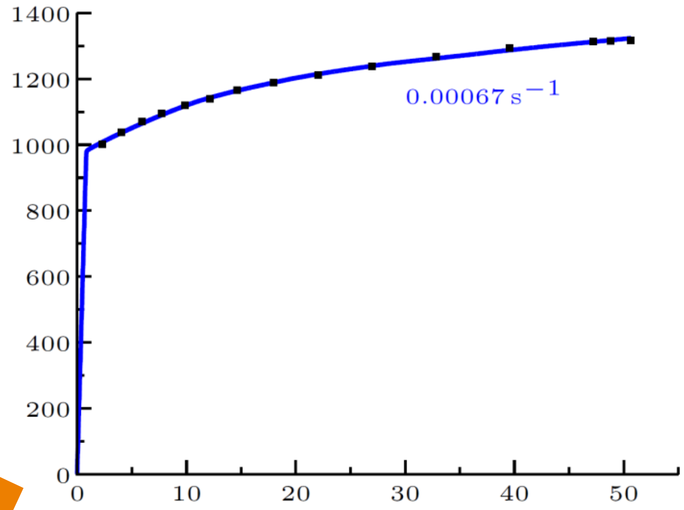
http://hdl.handle.net/1854/LU-4388117

Research

- Geometric algebra
- Grain growth
- Crystal plasticity



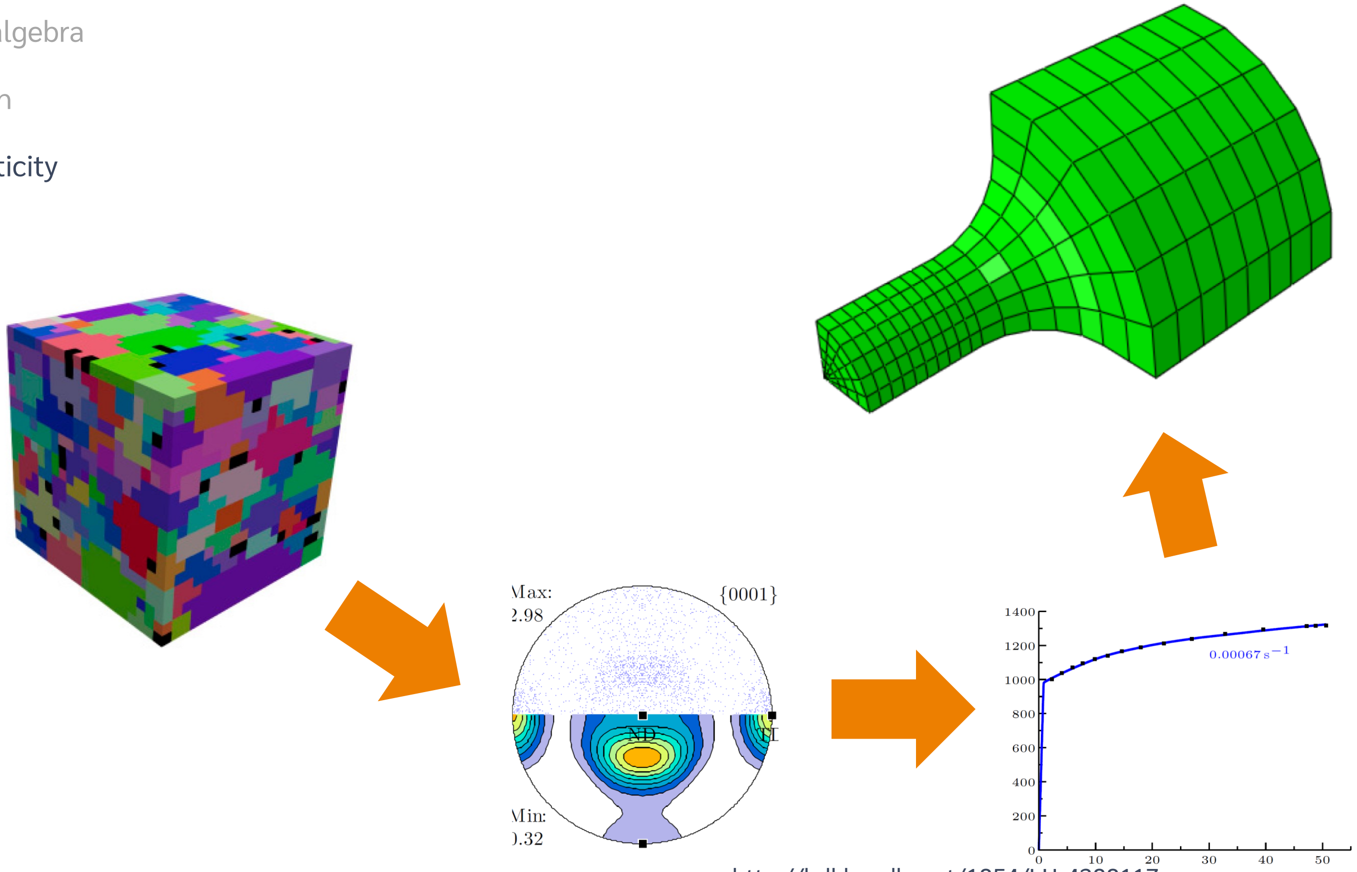




http://hdl.handle.net/1854/LU-4388117

Research

- Geometric algebra
- Grain growth
- Crystal plasticity



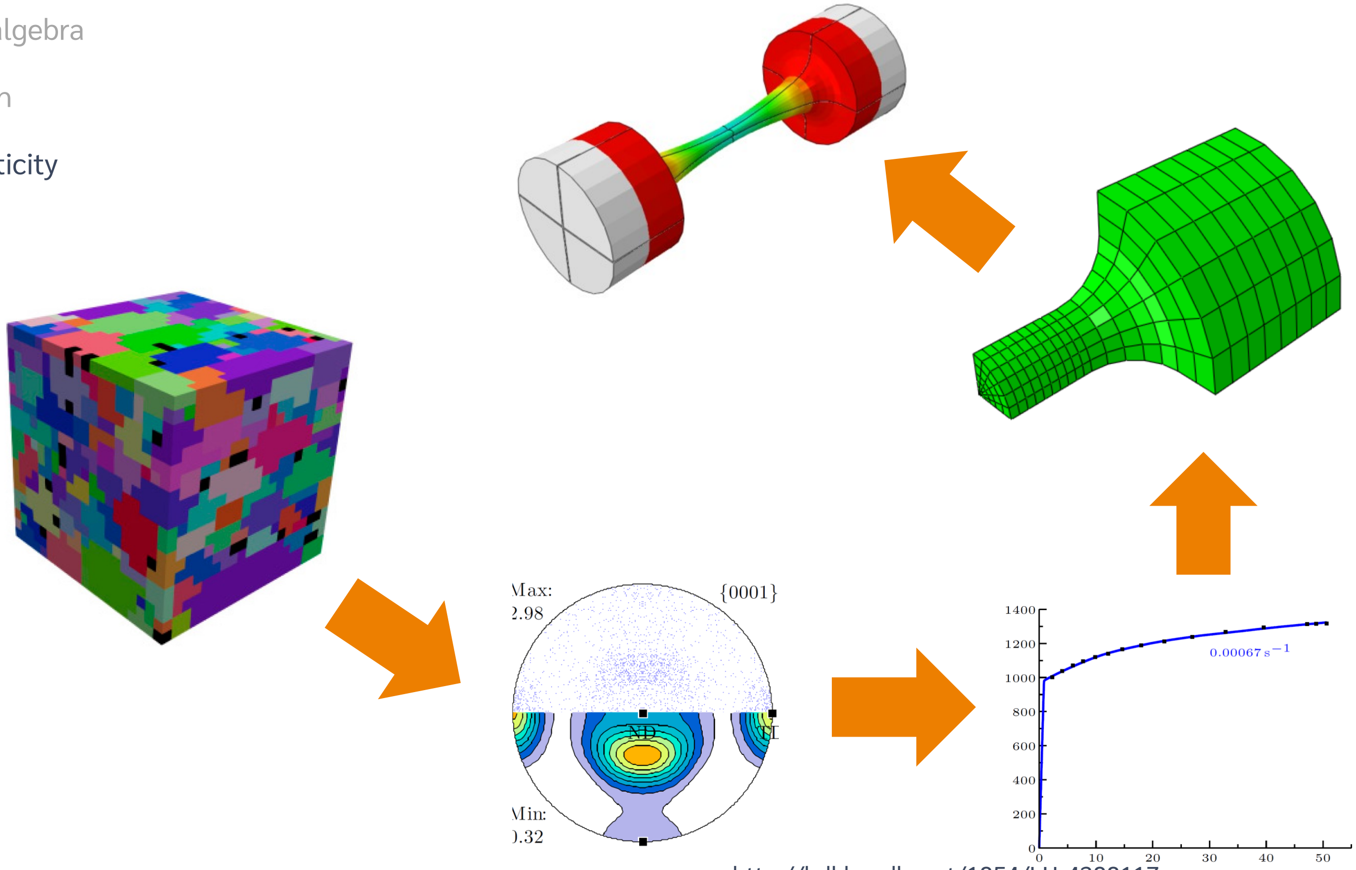
http://hdl.handle.net/1854/LU-4388117

jgl@dyalog.com

jesus.galanlopez@ugent.be

Research

- Geometric algebra
- Grain growth
- Crystal plasticity



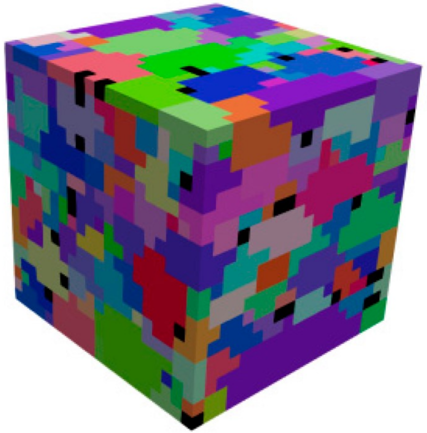
http://hdl.handle.net/1854/LU-4388117

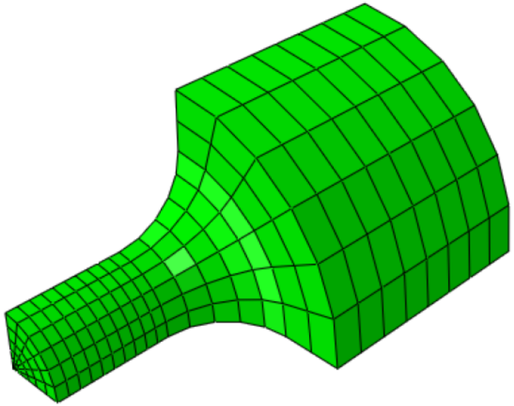
jgl@dyalog.com

jesus.galanlopez@ugent.be

Research

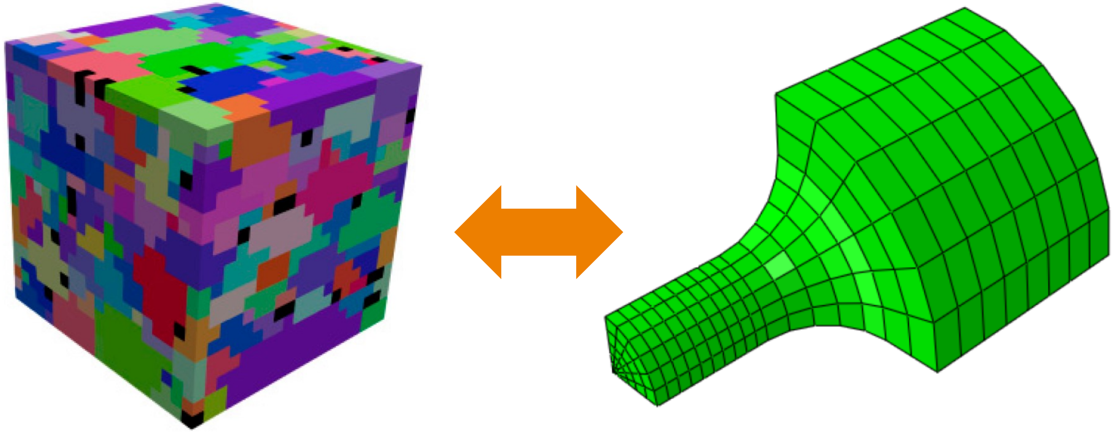
- Geometric algebra
- Grain growth
- Crystal plasticity





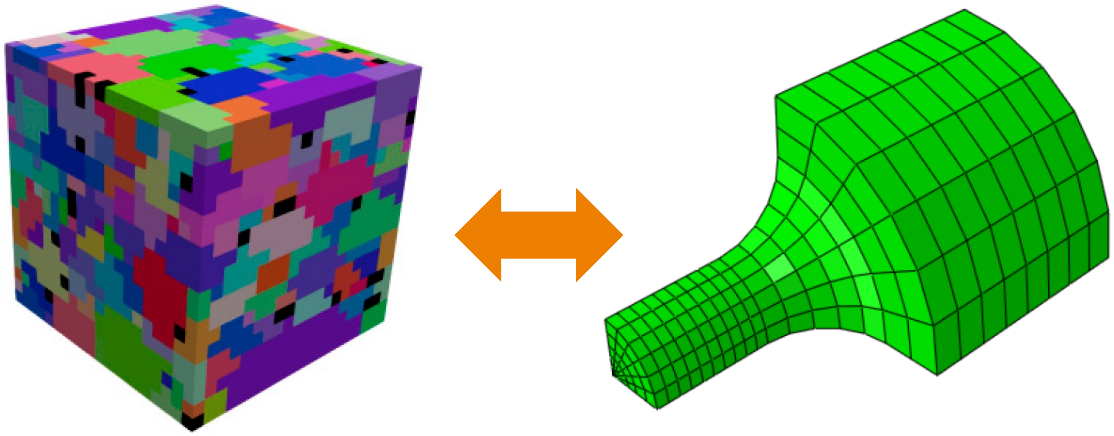
Research

- Geometric algebra
- Grain growth
- Crystal plasticity



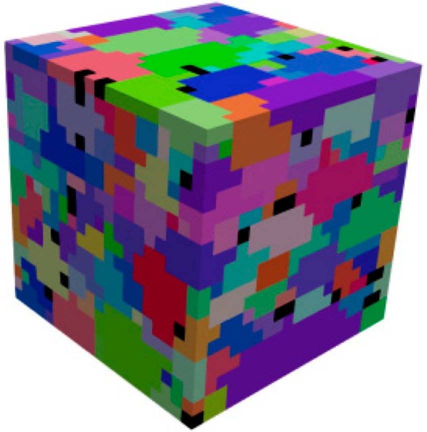
- Geometric algebra
- Grain growth
- Crystal plasticity

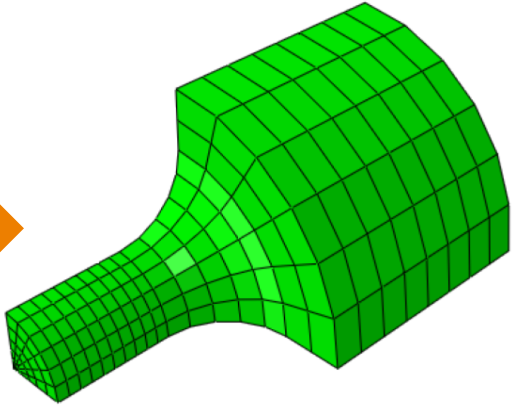
CPFEH



- Geometric algebra
- Grain growth
- Crystal plasticity

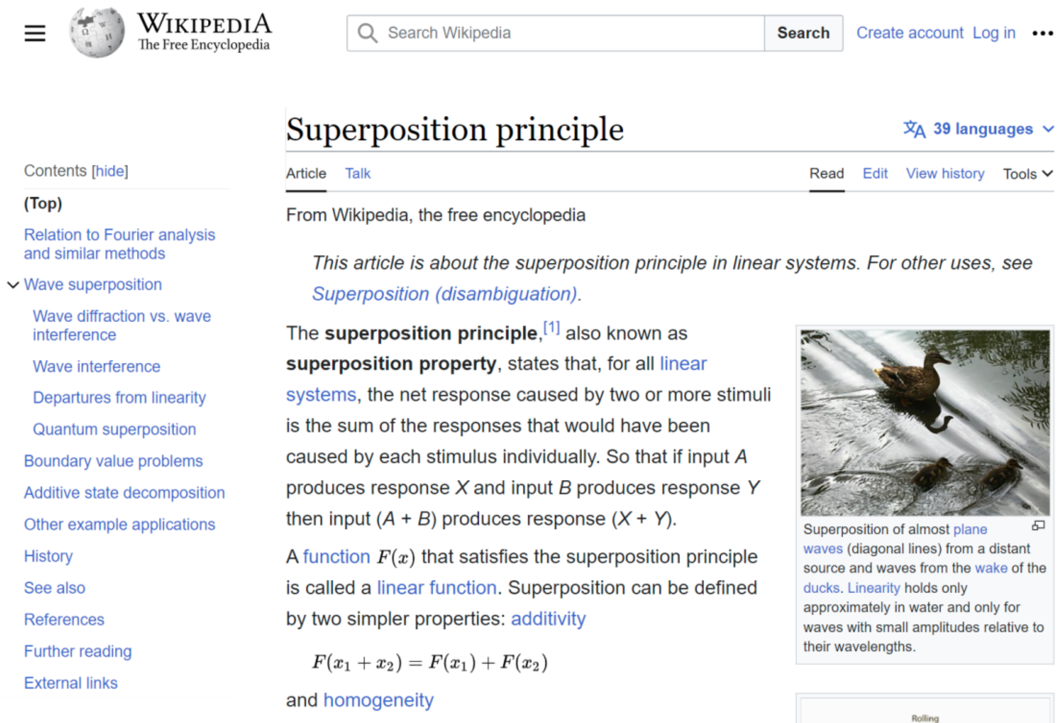
Crystal Plasticity Finite Element Homogenisation





- Geometric algebra
- Grain growth
- Crystal plasticity

Crystal Plasticity Finite Element Homogenisation



WIKIPEDIA The Free Encyclopedia

Search Wikipedia | Search | Create account | Log in

Superposition principle

39 languages

Article | Talk | Read | Edit | View history | Tools

Contents [hide]

From Wikipedia, the free encyclopedia

This article is about the superposition principle in linear systems. For other uses, see Superposition (disambiguation).

The **superposition principle**,[1] also known as **superposition property**, states that, for all linear systems, the net response caused by two or more stimuli is the sum of the responses that would have been caused by each stimulus individually. So that if input *A* produces response *X* and input *B* produces response *Y* then input (*A* + *B*) produces response (*X* + *Y*).

Superposition of almost plane waves (diagonal lines) from a distant source and waves from the wake of the ducks. Linearity holds only approximately in water and only for waves with small amplitudes relative to their wavelengths.

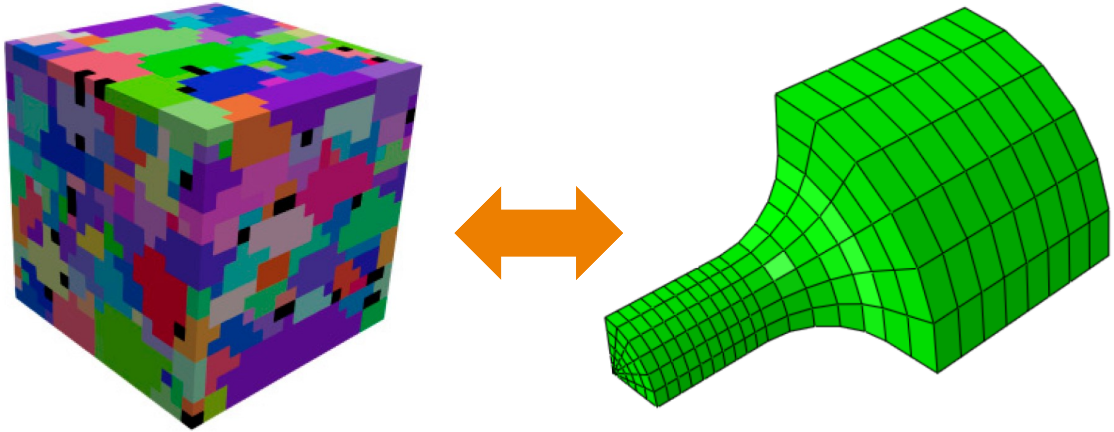
A function $F(x)$ that satisfies the superposition principle is called a linear function. Superposition can be defined by two simpler properties: additivity

$$F(x_1 + x_2) = F(x_1) + F(x_2)$$

and homogeneity

- Geometric algebra
- Grain growth
- Crystal plasticity

CPFEH←...

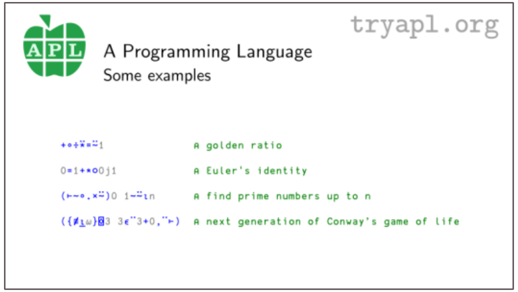


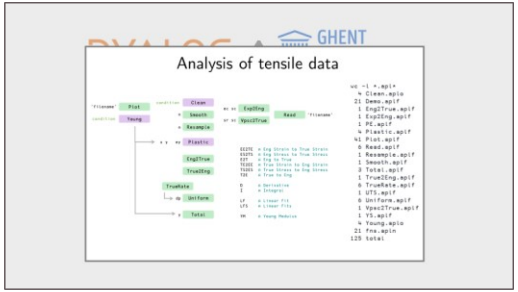
Research

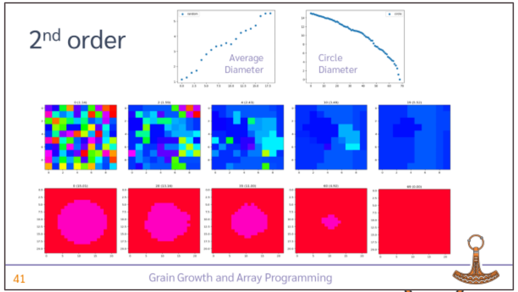
- Geometric algebra
- Grain growth
- Crystal plasticity

Dissemination and promotion

UNIVERSITEIT GENT

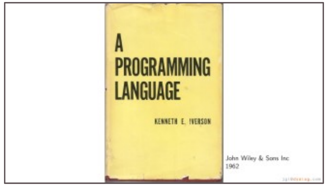


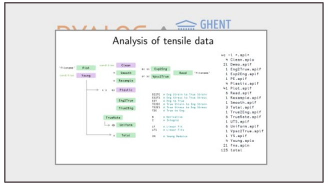


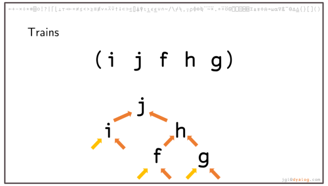


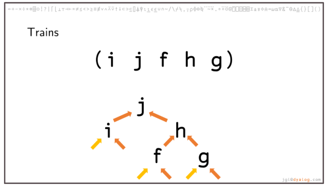
Dissemination and promotion

ELTE
EÖTVÖS LORÁND
UNIVERSITY









Dissemination and promotion

Conclusions

- APL is an interesting choice for the solution of materials science and engineering problems
- Convincing other researchers is not easy
- But there is potential
- We will keep trying

Plans

- Apply feedback
- More tutorials and publish them
- Continue research
  - Geometric algebra, grain growth, crystal plasticity
  - Articles and conferences
- Networking (more universities)

Thank you

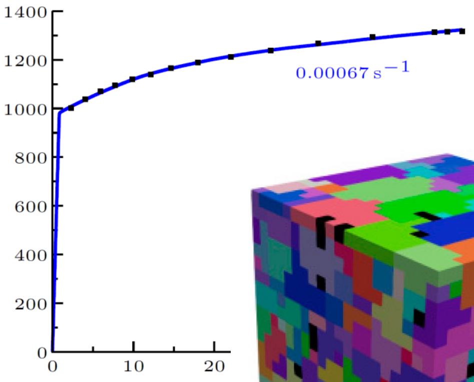
DYALOG

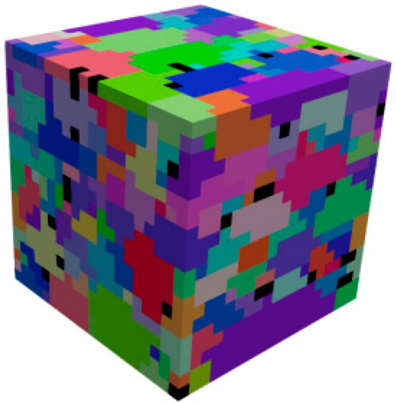
Elsinore 2023

APL and Metallurgy

```
TensileAnalysis ←
Crystallography ←
GrainGrowth ←
CrystalPlasticity ←
...
```

jgl@dyalog.com
jesus.galanlopez@ugent.be





GHENT UNIVERSITY