European Research Council erc

JG|U JOHANNES GUTENBERG UNIVERSITÄT MAINZ

INSTITUTE FOR GEOSCIENCES

GEOPHYSICS AND GEODYNAMICS

# A parallel modeling tool for lithospheric deformation

Anton Popov, Boris Kaus, Tobias Baumann, Adina Püsök, Georg Reuber

JGU-Mainz

Schloss Schney, Lichtenfels 2016

LaMEM Lithosphere and Mantle Evolution Model
FDSTAG Code

erc

European Research Council

JG|U

Johannes Gutenberg
Universität Mainz

JÜLICH
FORSCHUNGSZENTRUM


JUQUEEN - Jülich Blue Gene/Q

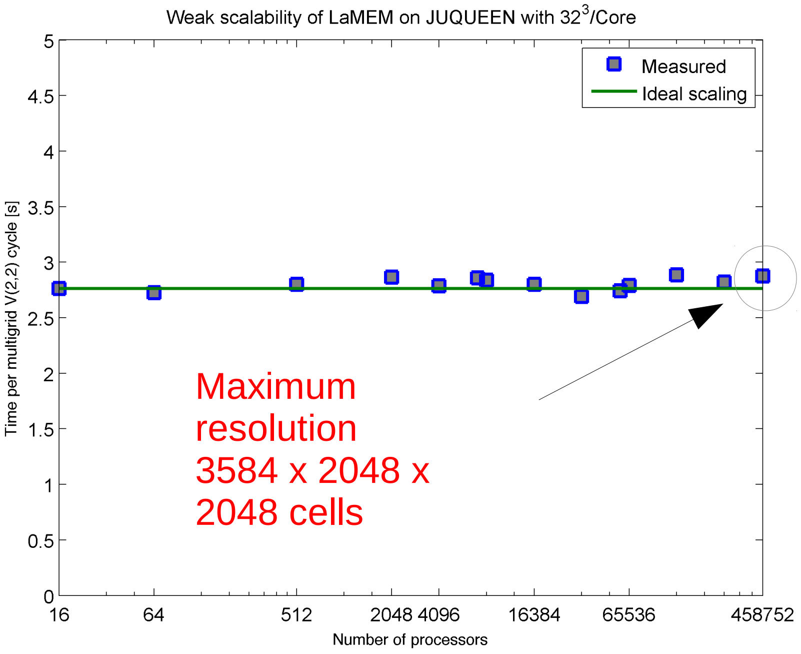
Argonne
NATIONAL LABORATORY

P Portable,
E Extensible
T Toolkit for
S Scientific
c Computation

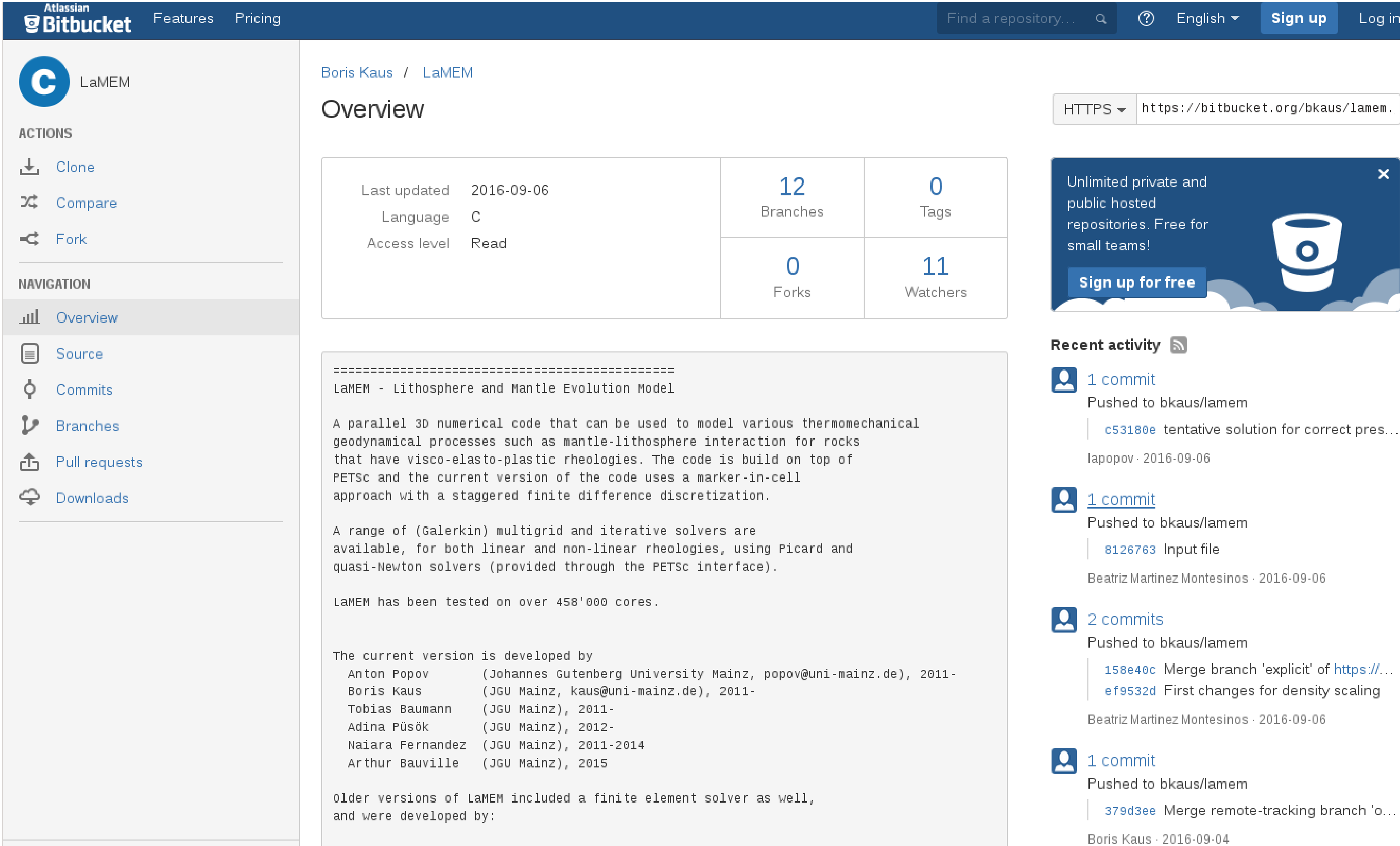
# Outline

# LaMEM (Lithosphere and Mantle Evolution Model)

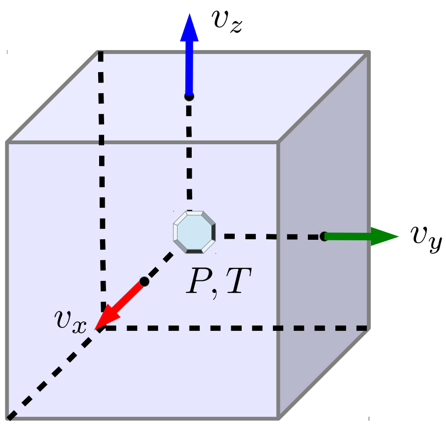
- 3D thermo-mechanical code, written in C
  uses PETSc.
- Nonlinear visco-elasto-plastic rheologies
- Runs on 1-458’752 processors
  routinely on 1024-4096
- Current version of code only supports staggered difference method (faster than FE)
- Can use a large variety of (multigrid) solvers (Galerkin GMG, AMG, Coupled/decoupled)
- Marker-and-cell method, free surface, (coupled to erosion model)
- Polygonal meshes to create (complex) input geometries.

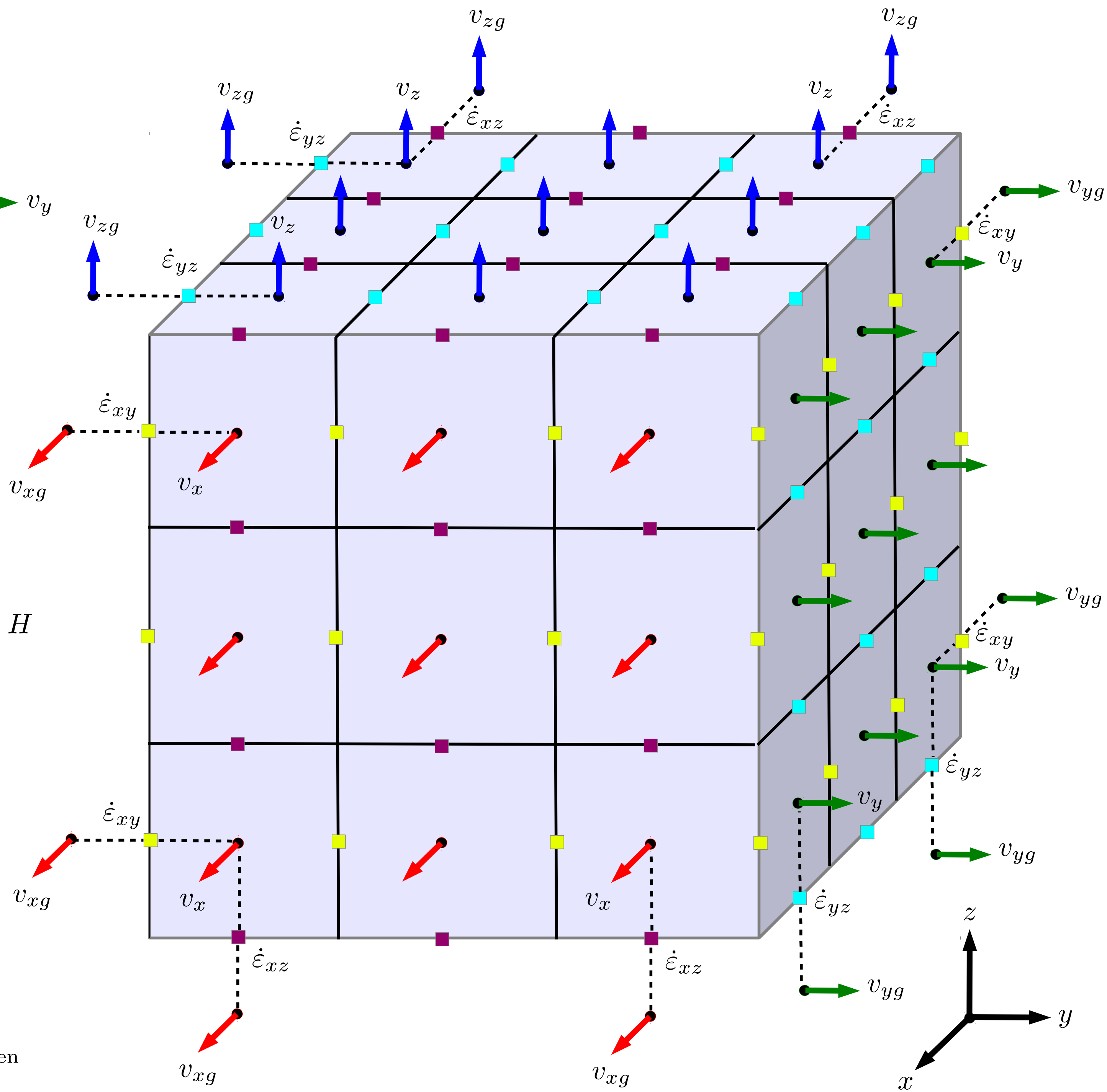



# https://bitbucket.org/bkaus/lamem



# Parallel staggered grid layout implementation



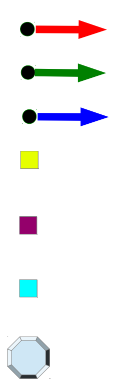

Equations:

$$\frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial P}{\partial x_i} + \rho g_i = 0$$

$$\frac{1}{\rho}\frac{D\rho}{Dt} + \frac{\partial v_i}{\partial x_i} = 0$$

$$\rho C_p \frac{DT}{Dt} = \frac{\partial}{\partial x_i}\left(\lambda \frac{\partial T}{\partial x_i}\right) + H$$

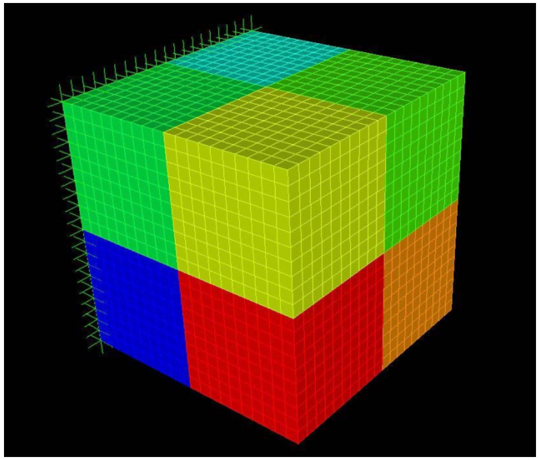
Legend:



- $v_x, q_x$
- $v_y, q_y$
- $v_z, q_z$
- $\dot{\varepsilon}_{xy}, \tau_{xy}, \eta_{xy}$
- $\dot{\varepsilon}_{xz}, \tau_{xz}, \eta_{xz}$
- $\dot{\varepsilon}_{yz}, \tau_{yz}, \eta_{yz}$
- $P, \dot{\varepsilon}_{xx}, \dot{\varepsilon}_{yy}, \dot{\varepsilon}_{zz}$
  $\tau_{xx}, \tau_{yy}, \tau_{zz}, \eta_{\text{cen}}$
  $T, \rho, C_p, \lambda, H$

# Parallel staggered grid layout implementation

## PETSc Distributed Array (DMDA)



- Parallel (MPI) each processor owns its local part of the grid
- Natural I-J-K indexing (global indices)!
- Local vectors with ghost points
- Boundary ghost points
- Global distributed vectors w/o ghost points
- Local to Global scatter/assembly operations

Altogether 8 DMDA objects are used

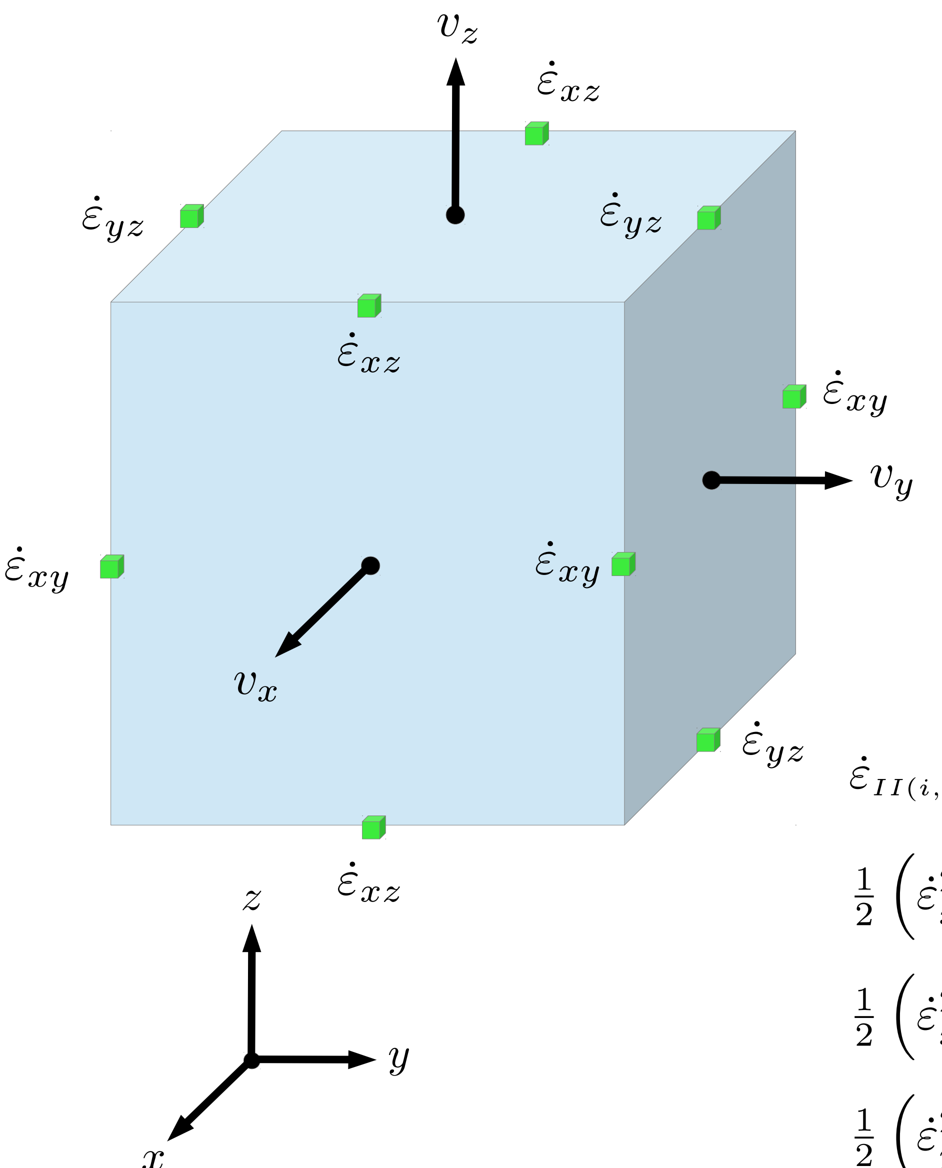
Corner DMDA is used for ParaView output

Coordinates are stored in local 1D arrays since grid is rectilinear

| NAME | x-size | y-size | z-size | Ghost points |
|---|---|---|---|---|
| DA_CORNER | Nx | Ny | Nz | None |
| DA_CENTER | Nx-1 | Ny-1 | Nz-1 | All |
| DA_XY | Nx | Ny | Nz-1 | None |
| DA_XZ | Nx | Ny-1 | Nz | None |
| DA_YZ | Nx-1 | Ny | Nz | None |
| DA_X | Nx | Ny-1 | Nz-1 | Y & Z |
| DA_Y | Nx-1 | Ny | Nz-1 | X & Z |
| DA_Z | Nx-1 | Ny-1 | Nz | X & Y |

# Nonlinear terms discretization

Example: interpolation scheme for the central points




There are three set of edge points: XY, XZ and YZ, defining corresponding shear strain rates:

$\dot{\varepsilon}_{xy},\ \dot{\varepsilon}_{xz},\ \dot{\varepsilon}_{yz}$

Central point defines normal strain rates:

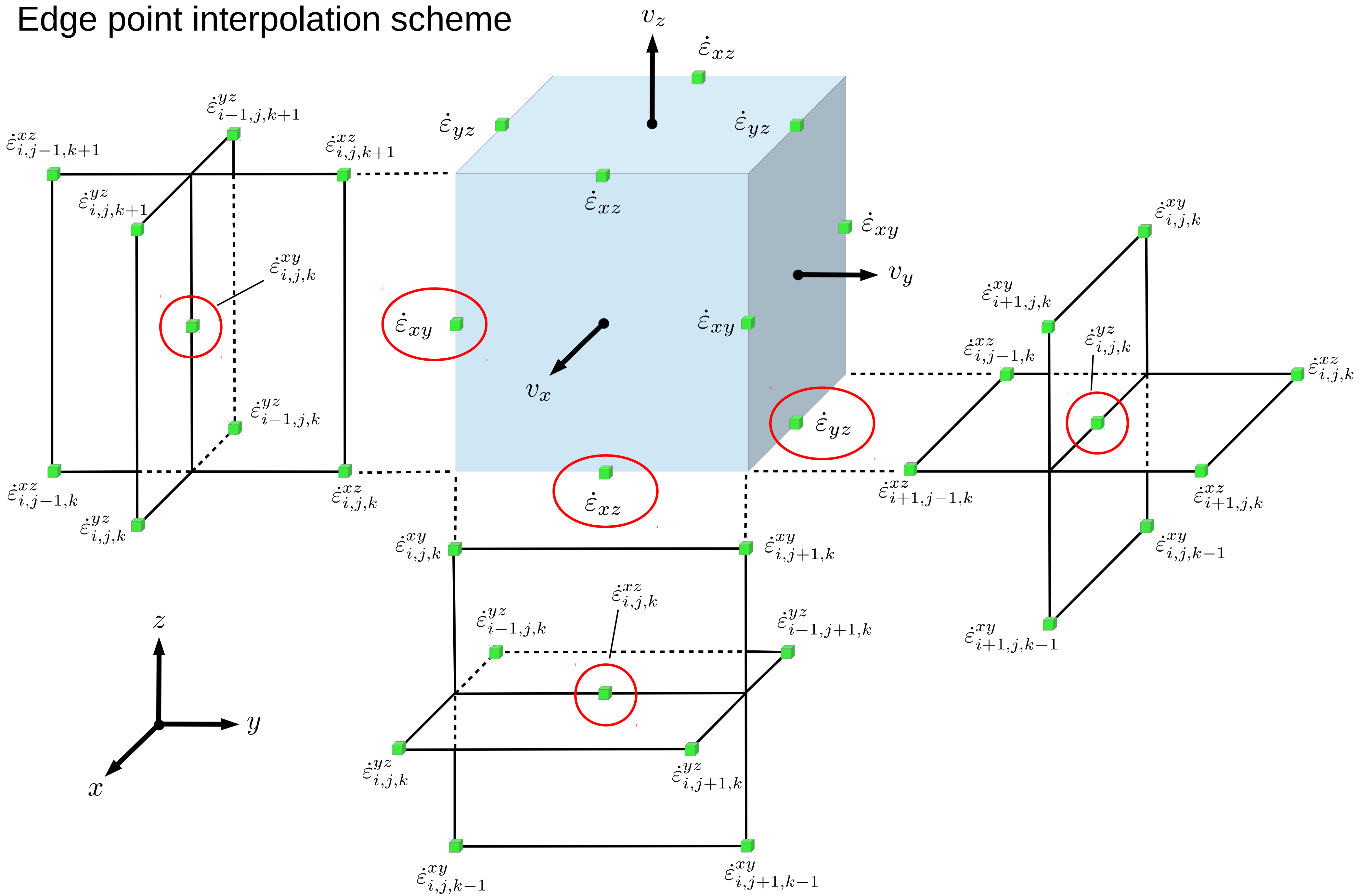
$\dot{\varepsilon}_{xx},\ \dot{\varepsilon}_{yy},\ \dot{\varepsilon}_{zz}$

Second invariant discretization:

$$\dot{\varepsilon}_{II(i,j,k)} = \dot{\varepsilon}^2_{xx(i,j,k)} + \dot{\varepsilon}^2_{yy(i,j,k)} + \dot{\varepsilon}^2_{zz(i,j,k)} +$$
$$\frac{1}{2}\left(\dot{\varepsilon}^2_{xy(i,j,k)} + \dot{\varepsilon}^2_{xy(i,j+1,k)} + \dot{\varepsilon}^2_{xy(i+1,j,k)} + \dot{\varepsilon}^2_{xy(i+1,j+1,k)}\right) +$$
$$\frac{1}{2}\left(\dot{\varepsilon}^2_{xz(i,j,k)} + \dot{\varepsilon}^2_{xz(i,j,k+1)} + \dot{\varepsilon}^2_{xz(i+1,j,k)} + \dot{\varepsilon}^2_{xz(i+1,j,k+1)}\right) +$$
$$\frac{1}{2}\left(\dot{\varepsilon}^2_{yz(i,j,k)} + \dot{\varepsilon}^2_{yz(i,j,k+1)} + \dot{\varepsilon}^2_{yz(i,j+1,k)} + \dot{\varepsilon}^2_{yz(i,j+1,k+1)}\right)$$

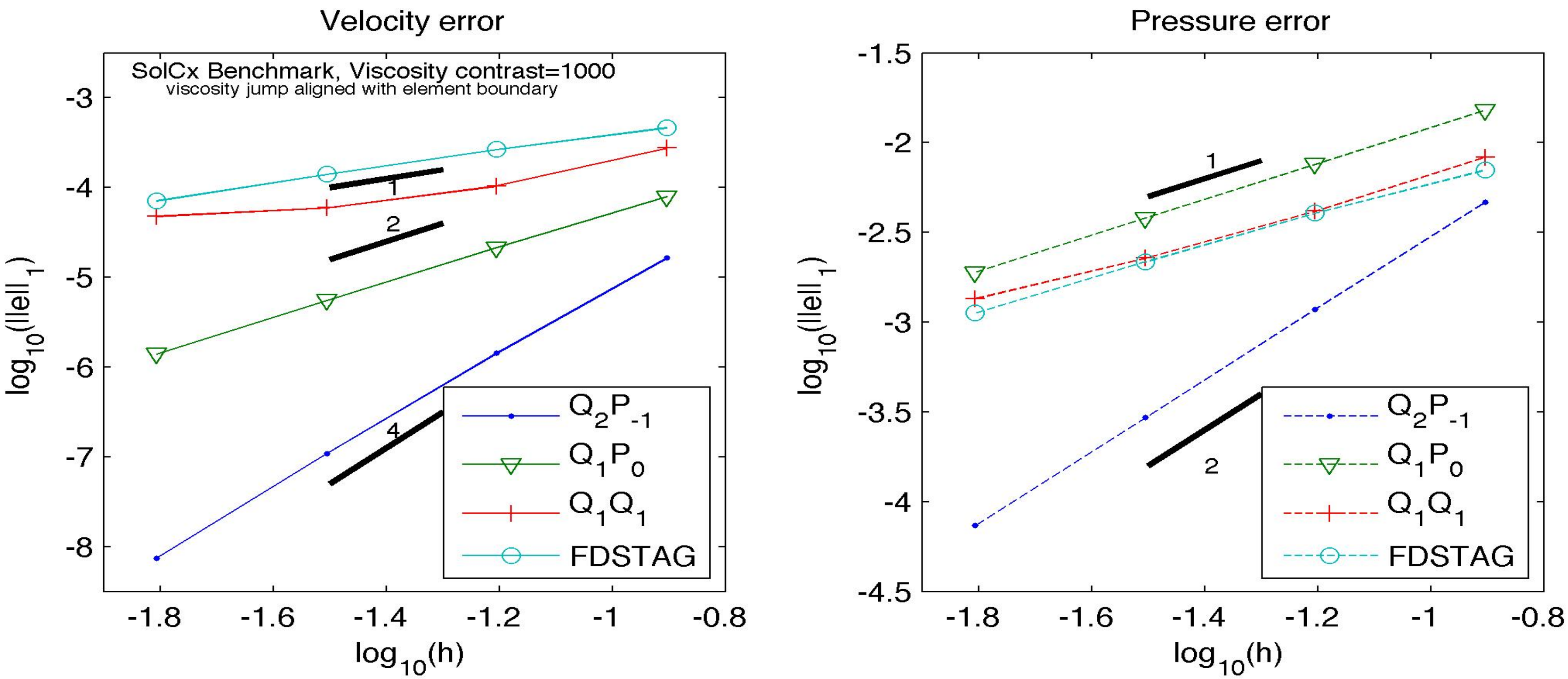
# Nonlinear terms discretization

Edge point interpolation scheme



# FDSTAG vs. FE convergence (SolCx benchmark)

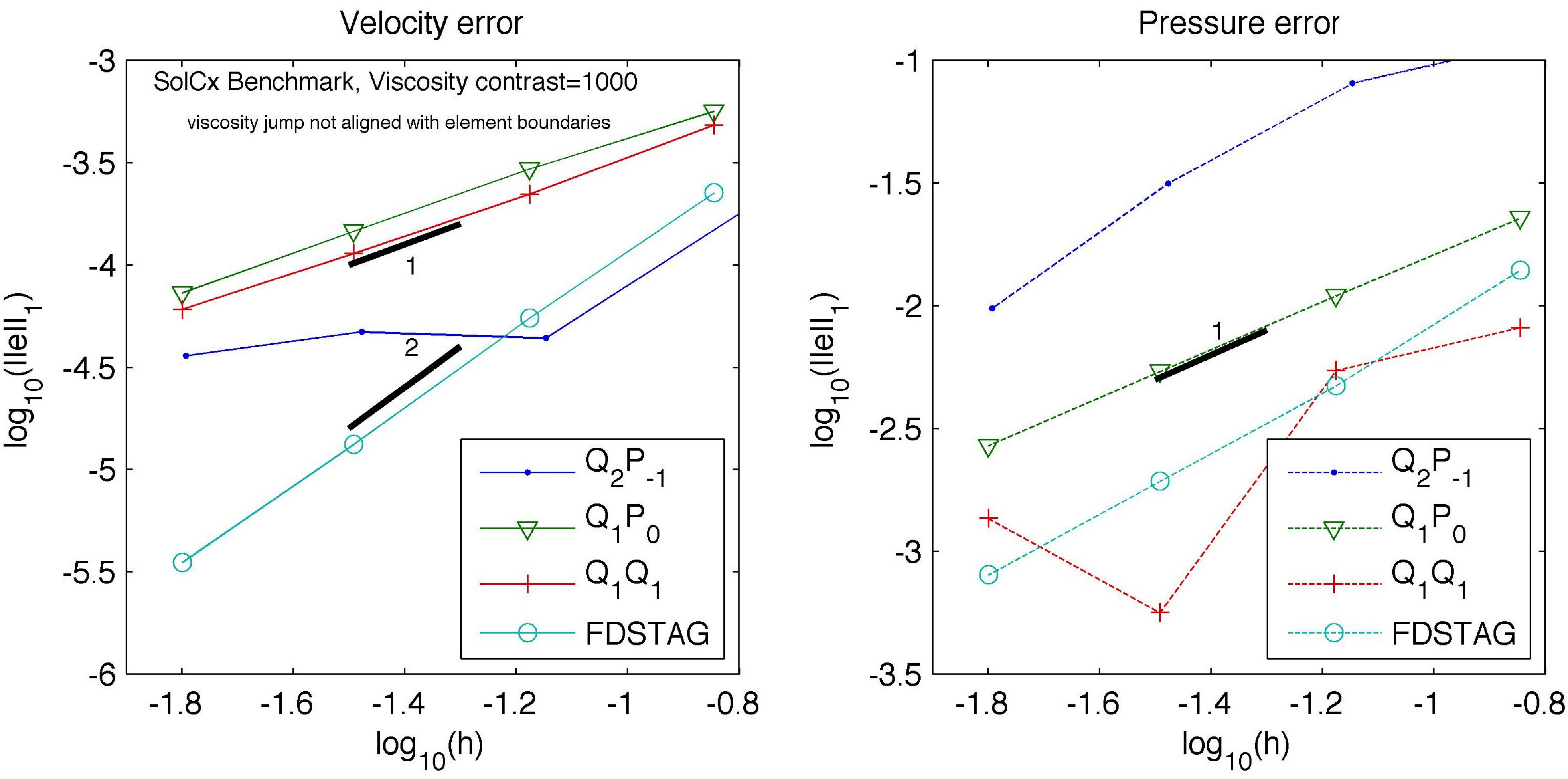
Viscosity contrast 1000, element boundary aligned with jump




High-order element wins

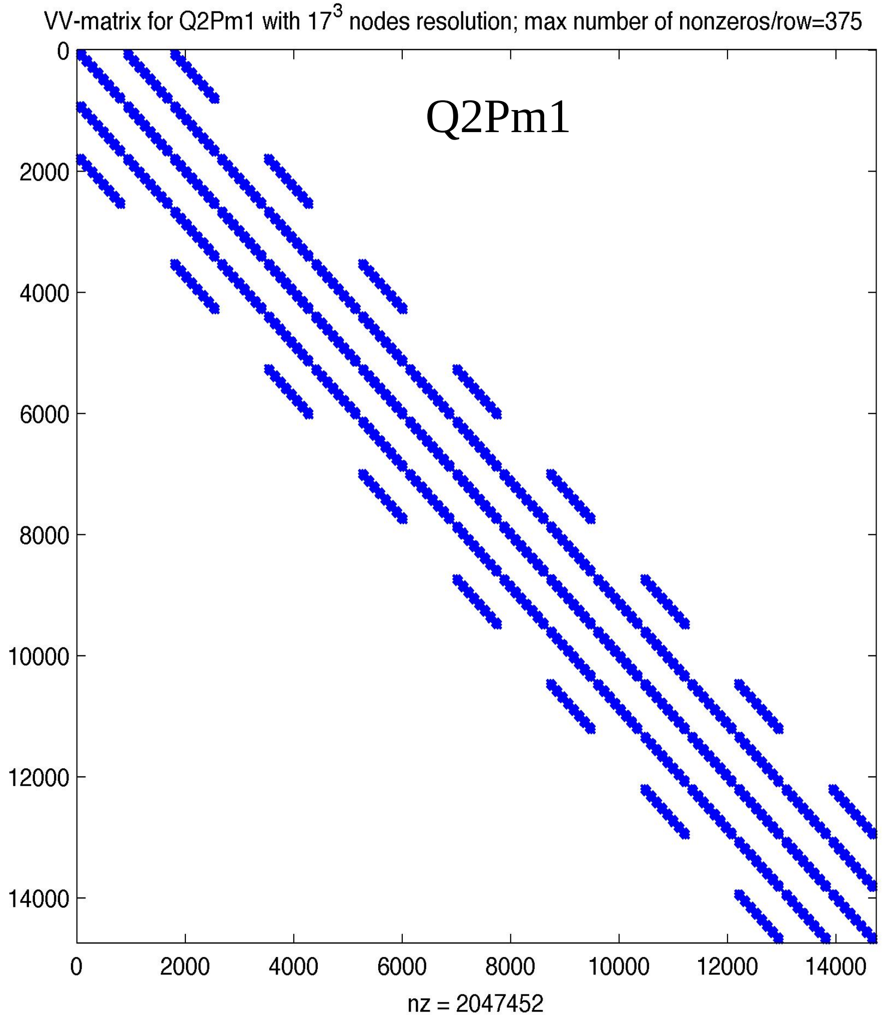
# FDSTAG vs. FE convergence (SolCx benchmark)

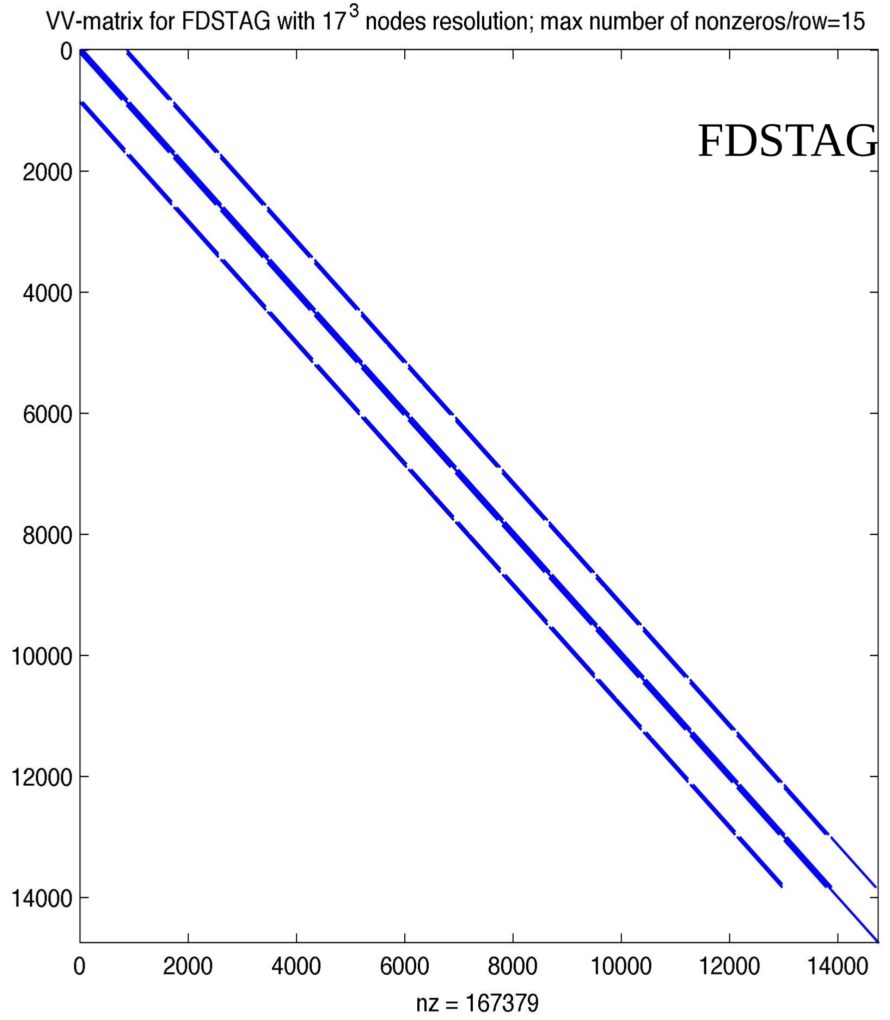
Viscosity contrast 1000, element boundary NOT aligned with jump




FDSTAG is not so bad
High-order element fails

# FDSTAG vs. FE memory footprint




FDSTAG requires *significantly* less memory!
Matrix-vector multiplications are much faster!

# Nonlinear viso-elasto-plastic rheology

Stress update

$$\tau_{ij} = 2\eta^* \dot{\varepsilon}^*_{ij}$$

Effective strain rate and invariant

$$\dot{\varepsilon}^*_{ij} = \dot{\varepsilon}_{ij} + \frac{\tau^*_{ij}}{2G\Delta t} \qquad \dot{\varepsilon}^*_{II} = \left(\tfrac{1}{2}\dot{\varepsilon}^*_{ij}\dot{\varepsilon}^*_{ij}\right)^{1/2}$$

Stress rotation terms (incremental stress rotation as described later)

$$\tau^*_{ij} = \tau^n_{ij} + \Delta t \left(w_{ik}\tau^n_{kj} - \tau^n_{ik}w_{kj}\right)$$

Deviatoric strain rate

$$\dot{\varepsilon}_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial v_k}{\partial x_k}\delta_{ij}$$

Spin tensor

$$\omega_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} - \frac{\partial v_j}{\partial x_i}\right)$$

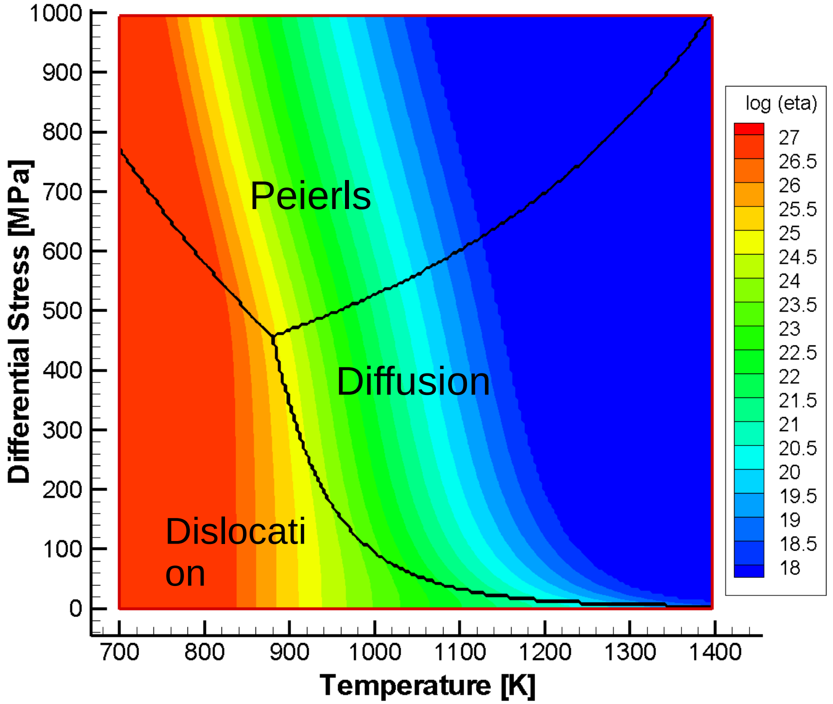
Effective viscosity

$$\eta^* = \min\left[\left(\tfrac{1}{G\Delta t} + \tfrac{1}{\eta_l} + \tfrac{1}{\eta_n} + \tfrac{1}{\eta_p}\right)^{-1},\ \tfrac{\tau_Y}{2\dot{\varepsilon}^*_{II}}\right]$$

# Nonlinear viso-elasto-plastic rheology

Drucker-Prager yield stress

$$\tau_Y = \mu P + c$$




Diffusion Dislocation and Peierls constants

$$A_l = B_l \ \exp\left[-\frac{E_l+pV_l}{RT}\right],$$
$$A_n = B_n \ \exp\left[-\frac{E_n+pV_n}{RT}\right]$$
$$A_p = \frac{B_p}{(\gamma\tau_p)^s} \ \exp\left[-\frac{E_p+pV_p}{RT}\,(1-\gamma)^q\right]$$

Effective creep viscosities

$$\eta_l = \tfrac{1}{2}\,(A_l)^{-1}$$
$$\eta_n = \tfrac{1}{2}\,(A_n)^{-\frac{1}{n}}\,(\dot{\varepsilon}^*_{II})^{\frac{1}{n}-1}$$
$$\eta_p = \tfrac{1}{2}\,(A_p)^{-\frac{1}{s}}\,(\dot{\varepsilon}^*_{II})^{\frac{1}{s}-1}$$

Peierls effective exponent

$$s = \frac{E_p + PV_p}{RT}\,(1-\gamma)^{q-1}\,q\,\gamma$$

# Global nonlinear iterations

Preconditioned Newton iteration with line search

$$P^{-1}J\left(x_k\right)\delta x_k = -P^{-1}r\left(x_k\right) \qquad x_{k+1} = x_k + \alpha\,\delta x_k$$

Jacobian and residual

$$J = \begin{pmatrix} \frac{\partial r_M}{\partial v} & \frac{\partial r_M}{\partial P} & \frac{\partial r_M}{\partial T} \\ \frac{\partial r_C}{\partial v} & \frac{\partial r_C}{\partial P} & \frac{\partial r_C}{\partial T} \\ \frac{\partial r_E}{\partial v} & \frac{\partial r_E}{\partial P} & \frac{\partial r_E}{\partial T} \end{pmatrix} \qquad r = \begin{pmatrix} r_M \\ r_C \\ r_E \end{pmatrix} \qquad x = \begin{pmatrix} v \\ P \\ T \end{pmatrix}$$

$r$: Momentum, Continuity, Energy

$x$: Velocity, Pressure, Temperature

Jacobian-Free-Newton-Krylov (**JFNK**) or analytical matrix-free Jacobian

$$Jy \approx \frac{r\left(x+hy\right) - r\left(x\right)}{h}$$

`-snes_mf_operator` (PETSc **SNES** solver)

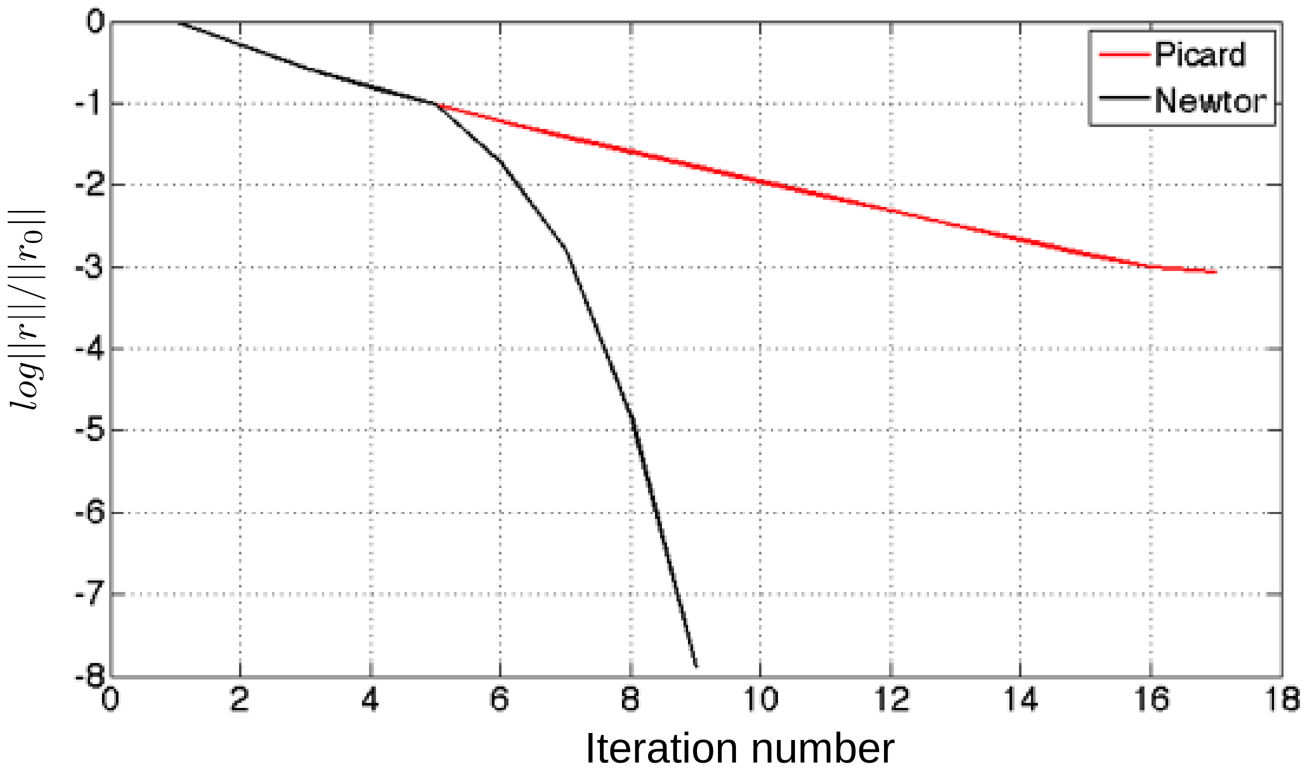
Preconditioner matrix

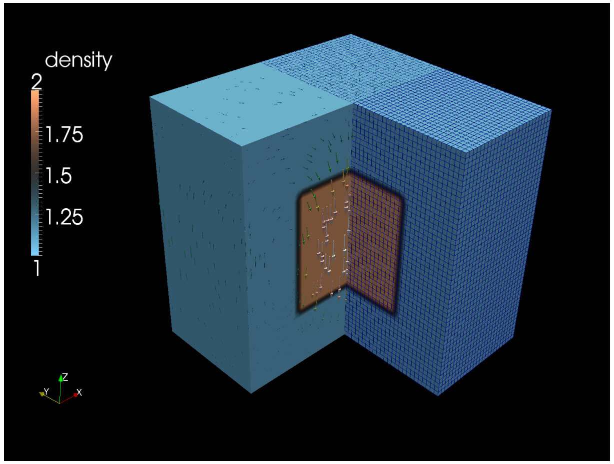
$$P = \begin{pmatrix} K & G & 0 \\ D & C & 0 \\ 0 & 0 & E \end{pmatrix}$$

# Picard vs. Newton

Quasi-linear residual form: $r\left(x\right)=A\left(x\right)x-b=0$

Picard fixed-point iteration: $J\left(x\right)\approx A\left(x\right)$




Picard approximation facilitates convergence at the initial stages

Switching to Picard can improve Newton convergence

# Analytical Jacobian Finite Elements

## Nonlinear residuals

$$r^U = \int_\Omega B^T(\tau - mP) - N_U^T \rho g \, d\Omega$$

$$r^P = -\int_\Omega N_P^T m^T \dot{\varepsilon} \, d\Omega$$

## Jacobian iteration

$$\begin{bmatrix} \Delta U_k \\ \Delta P_k \end{bmatrix} = -\begin{bmatrix} J^{UU} & J^{UP} \\ J^{PU} & 0 \end{bmatrix}^{-1} \begin{bmatrix} r_k^U \\ r_k^P \end{bmatrix}$$

$$\begin{bmatrix} U_{k+1} \\ P_{k+1} \end{bmatrix} = \begin{bmatrix} U_k \\ P_k \end{bmatrix} + \alpha \begin{bmatrix} \Delta U_k \\ \Delta P_k \end{bmatrix}$$

## Stress and strain rate vectors

$$\begin{aligned}
\dot{\varepsilon} &= [\dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy}\dot{\varepsilon}_{zz}\dot{\gamma}_{xy}\dot{\gamma}_{xz}\dot{\gamma}_{yz}]^T \\
\underline{\dot{\varepsilon}} &= [\dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy}\dot{\varepsilon}_{zz}\dot{\varepsilon}_{xy}\dot{\varepsilon}_{yx}\dot{\varepsilon}_{xz}\dot{\varepsilon}_{zx}\dot{\varepsilon}_{yz}\dot{\varepsilon}_{zy}]^T \\
\tau &= [\tau_{xx}\tau_{yy}\tau_{zz}\tau_{xy}\tau_{xz}\tau_{yz}]^T \\
\underline{\tau} &= [\tau_{xx}\tau_{yy}\tau_{zz}\tau_{xy}\tau_{yx}\tau_{xz}\tau_{zx}\tau_{yz}\tau_{zy}]^T
\end{aligned}$$

## Projection matrix and projection

$$Q = \frac{1}{2}\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\tau = Q\underline{\tau}, \quad \underline{\dot{\varepsilon}} = Q^T \dot{\varepsilon}$$

## Pressure projection vector

$$\mathrm{m} = [1\,1\,1\,0\,0\,0\,0\,0\,0]^T$$

# Analytical Jacobian Finite Elements

Jacobian blocks

$$J^{UU} = \int_\Omega B^T Q\, T\, Q^T B \, d\Omega$$

$$J^{UP} = -\int_\Omega B^T \left(m - \mu Q \underline{q}\right) N_P \, d\Omega$$

$$J^{PU} = -\int_\Omega N_P^T m^T B \, d\Omega$$

Tangent matrix

$$T = 2\eta^* \left(I + \beta\, \underline{q}\, \underline{q}^T\right)$$

Visco-elastic nonlinear parameter

$$\beta = \frac{1}{2}\left(\frac{1}{n} - 1\right) \frac{\eta^*}{\eta_n} + \frac{1}{2}\left(\frac{1}{s} - 1\right) \frac{\eta^*}{\eta_p}$$

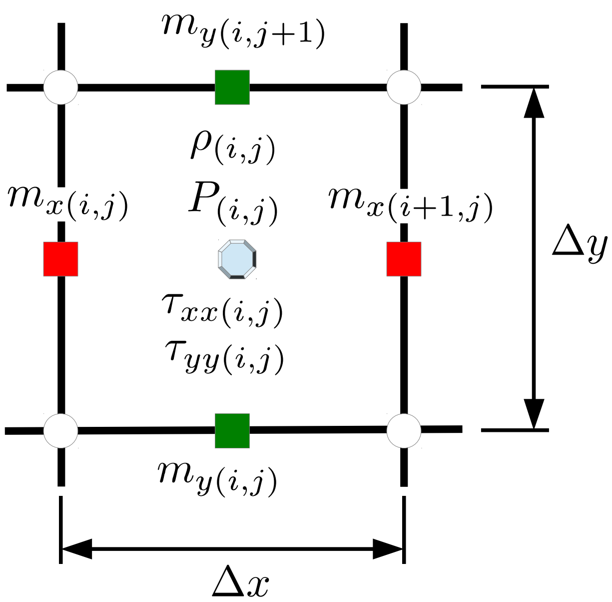
Plastic nonlinear parameter

$$\beta = -\frac{1}{2}$$

Normalized flow vector

$$\underline{q} = \frac{\underline{\dot{\varepsilon}}}{\dot{\varepsilon}_{II}^*}$$

# Analytical Jacobian Finite Difference (example)




## Momentum residual contributions

$$\Delta m_{x(i,j)} = + \frac{\tau_{xx(i,j)} - P_{(i,j)}}{\Delta x} + \frac{1}{2}\rho_{(i,j)} g_x$$

$$\Delta m_{x(i+1,j)} = - \frac{\tau_{xx(i,j)} - P_{(i,j)}}{\Delta x} + \frac{1}{2}\rho_{(i,j)} g_x$$

## Residual derivatives (velocity)

$$\frac{\partial \Delta m_{x(i,j)}}{\partial \boldsymbol{v}} = + \frac{1}{\Delta x}\frac{\partial \tau_{xx(i,j)}}{\partial \boldsymbol{v}}$$

$$\frac{\partial \Delta m_{x(i+1,j)}}{\partial \boldsymbol{v}} = - \frac{1}{\Delta x}\frac{\partial \tau_{xx(i,j)}}{\partial \boldsymbol{v}}$$

## Residual derivatives (pressure)

$$\frac{\partial \Delta m_{x(i,j)}}{\partial P} = + \frac{1}{\Delta x}\frac{\partial \tau_{xx(i,j)}}{\partial P} - \frac{1}{\Delta x}$$

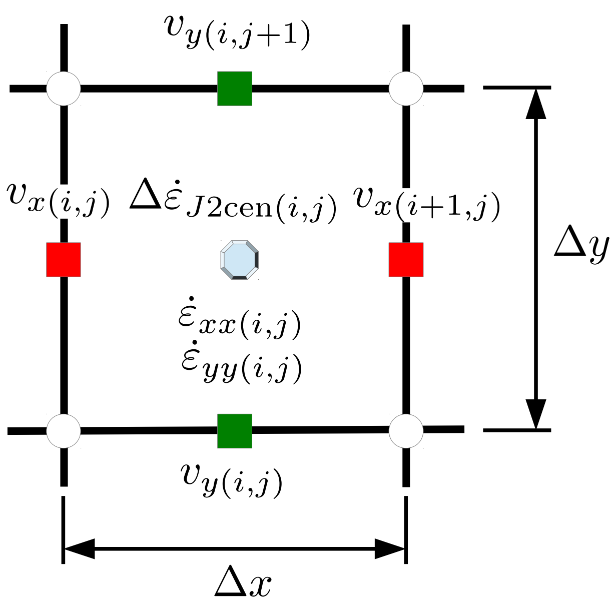
$$\frac{\partial \Delta m_{x(i+1,j)}}{\partial P} = - \frac{1}{\Delta x}\frac{\partial \tau_{xx(i,j)}}{\partial P} + \frac{1}{\Delta x}$$

## Stress derivatives

$$\frac{\partial \tau_{xx(i,j)}}{\partial \boldsymbol{v}} = 2\frac{\partial \eta^*_{\mathrm{cen}(i,j)}}{\partial \boldsymbol{v}}\dot{\varepsilon}^*_{xx(i,j)} + 2\eta^*_{\mathrm{cen}(i,j)}\frac{\dot{\varepsilon}^*_{xx(i,j)}}{\partial \boldsymbol{v}}$$

$$\frac{\partial \tau_{xx(i,j)}}{\partial P} = \mu\frac{\dot{\varepsilon}^*_{xx(i,j)}}{\dot{\varepsilon}^*_{II}}$$

# Analytical Jacobian Finite Difference (example)




## Viscosity derivatives

$$\frac{\partial \eta^*}{\partial \boldsymbol{v}} = \frac{\partial \eta^*}{\partial \dot{\varepsilon}_{II}} \frac{\partial \dot{\varepsilon}_{II}}{\partial \dot{\varepsilon}_{J2}} \frac{\partial \dot{\varepsilon}_{J2}}{\partial \boldsymbol{v}} \qquad \frac{\partial \dot{\varepsilon}_{II}}{\partial \dot{\varepsilon}_{J2}} = \tfrac{1}{2} \left(\dot{\varepsilon}_{II}\right)^{-1}$$

$$\frac{\partial \eta^*}{\partial \dot{\varepsilon}_{II}} = \left(\frac{\eta^*}{\eta_n}\right)^2 \left(\frac{1}{n} - 1\right) \frac{\eta_n}{\dot{\varepsilon}_{II}^*} + \left(\frac{\eta^*}{\eta_p}\right)^2 \left(\frac{1}{s} - 1\right) \frac{\eta_p}{\dot{\varepsilon}_{II}^*}$$

$$\frac{\partial \eta^*}{\partial \dot{\varepsilon}_{II}} = -\frac{1}{2} \frac{\eta^*}{\dot{\varepsilon}_{II}^*}$$ (Plastic case)

## Strain rate derivatives

$$\dot{\varepsilon}_{xx(i,j)} = \frac{v_{x(i+1,j)} - v_{x(i,j)}}{\Delta x} \qquad \frac{\partial \dot{\varepsilon}_{yy(i,j)}}{\partial v_{y(i,j)}} = -\frac{1}{\Delta y} \qquad \frac{\partial \dot{\varepsilon}_{yy(i,j)}}{\partial v_{y(i,j+1)}} = \frac{1}{\Delta y}$$

## Second invariant contribution derivatives

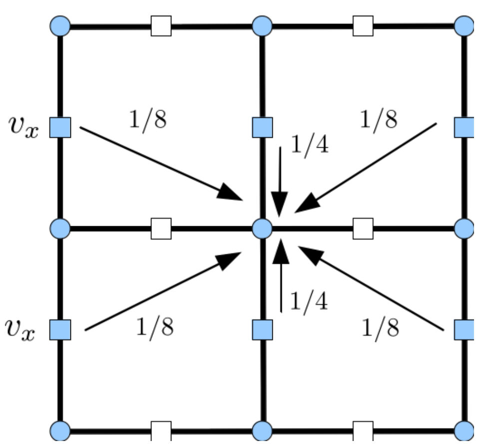
$$\Delta\dot{\varepsilon}_{J2\mathrm{cen}(i,j)} = \tfrac{1}{2}\left(\dot{\varepsilon}_{xx(i,j)}^2 + \dot{\varepsilon}_{yy(i,j)}^2\right)$$

$$\frac{\partial \Delta\dot{\varepsilon}_{J2\mathrm{cen}(i,j)}}{\partial v_{x(i,j)}} = -\frac{1}{\Delta x}\dot{\varepsilon}_{xx(i,j)} \qquad \frac{\partial \Delta\dot{\varepsilon}_{J2\mathrm{cen}(i,j)}}{\partial v_{x(i+1,j)}} = \frac{1}{\Delta x}\dot{\varepsilon}_{xx(i,j)}$$

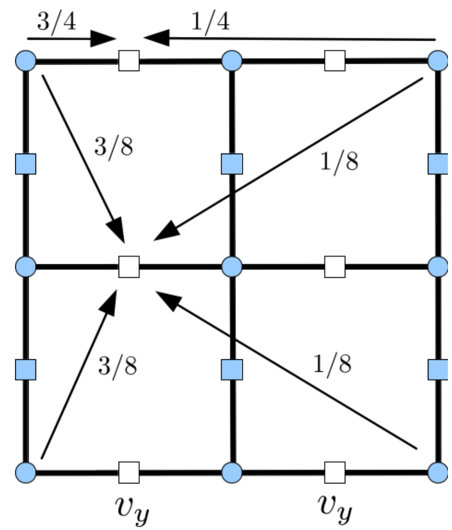
$$\frac{\partial \Delta\dot{\varepsilon}_{J2\mathrm{cen}(i,j)}}{\partial v_{y(i,j)}} = -\frac{1}{\Delta y}\dot{\varepsilon}_{yy(i,j)} \qquad \frac{\partial \Delta\dot{\varepsilon}_{J2\mathrm{cen}(i,j)}}{\partial v_{y(i,j+1)}} = \frac{1}{\Delta y}\dot{\varepsilon}_{yy(i,j)}$$

# Galerkin Multigrid

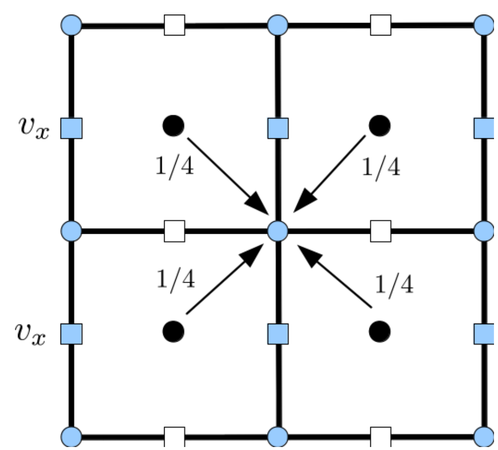
### Velocity restriction



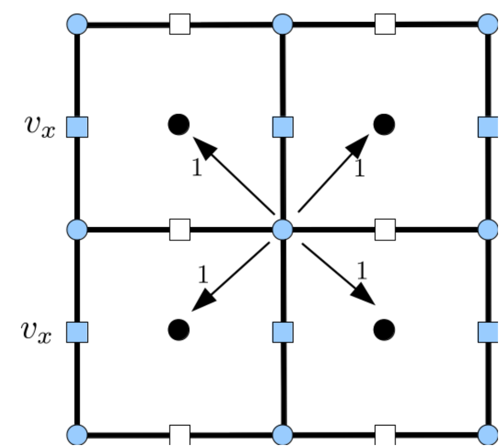

### Velocity prolongation




### Pressure restriction




### Pressure prolongation




Described in Cai et al. 2014

## Galerkin coarsening

$$\boldsymbol{r}_{coarse} = \boldsymbol{R}\,\boldsymbol{r}_{fine}$$ - Restriction

$$\boldsymbol{x}_{fine} = \boldsymbol{P}\,\boldsymbol{x}_{coarse}$$ - Prolongation

$$\boldsymbol{A}_{coarse} = \boldsymbol{R}\,\boldsymbol{A}_{fine}\,\boldsymbol{P}$$ - Coarsening

## Preconditioners

$$P_c = \begin{pmatrix} K & G \\ D & -\frac{1}{\eta} I \end{pmatrix}$$

(COUPLED)

Galerkin MG applied to full matrix

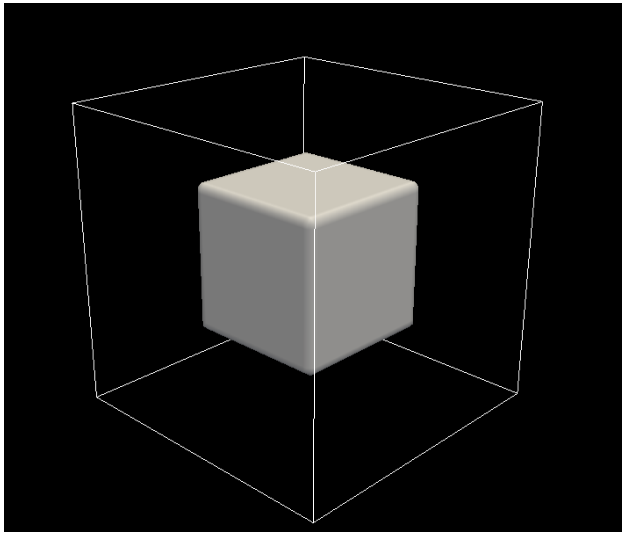
$$P_u = \begin{pmatrix} K & G \\ & -\frac{1}{\eta} I \end{pmatrix}$$

(BLOCK MG - uncoupled)

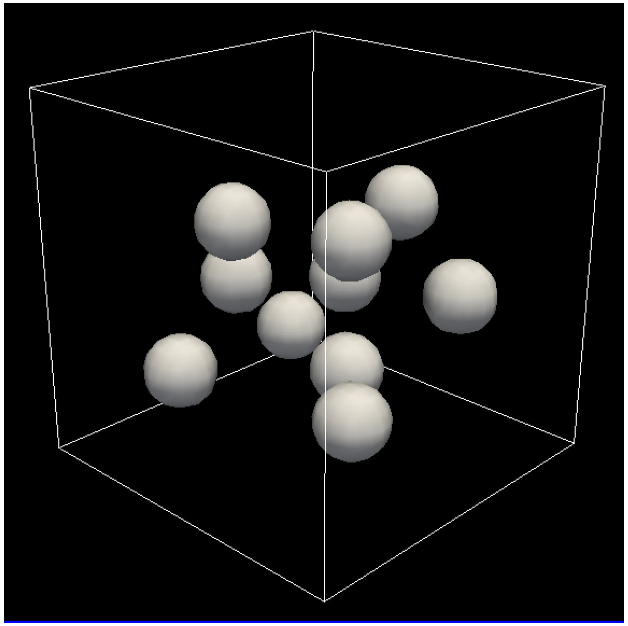
Galerkin MG applied to **K** block only

# Coupled vs. Uncoupled MG (1 vs. 10 falling blocks)

| 64³ nodes<br>3 GMG levels | Falling block setup | | | |
|---|---|---|---|---|
| | Coupled multigrid | | Block MG | |
| **Viscosity contrast** | *# outer KSP it* | *Time [s]* | *# outer KSP it* | *Time [s]* |
| 1 | 7 | 6.1 | 9 | 6.4 |
| 10 | 10 | 8.2 | 13 | 8.4 |
| 100 | 12 | 9.6 | 20 | 11.9 |
| 1000 | 17 | 13.2 | 30 | 17.1 |
| 10000 | 40 | 29 | 71 | 38 |
| 1.00E+05 | 155 | 107 | 267 | 137 |



| 64³ nodes<br>3 GMG levels | Multiple spheres setup | | | |
|---|---|---|---|---|
| | Coupled multigrid | | Block MG | |
| **Viscosity contrast** | *# outer KSP it* | *Time [s]* | *# outer KSP it* | *Total solve [s]* |
| 1 | 7 | 6.1 | 6 | 3.9 |
| 10 | 10 | 8 | 11 | 6 |
| 100 | 15 | 11.7 | 18 | 10.2 |
| 1000 | 36 | 27.6 | 45 | 23 |
| 10000 | 114 | 82 | 154 | 80.5 |
| 1.00E+05 | 378 | 266 | 585 | 297 |



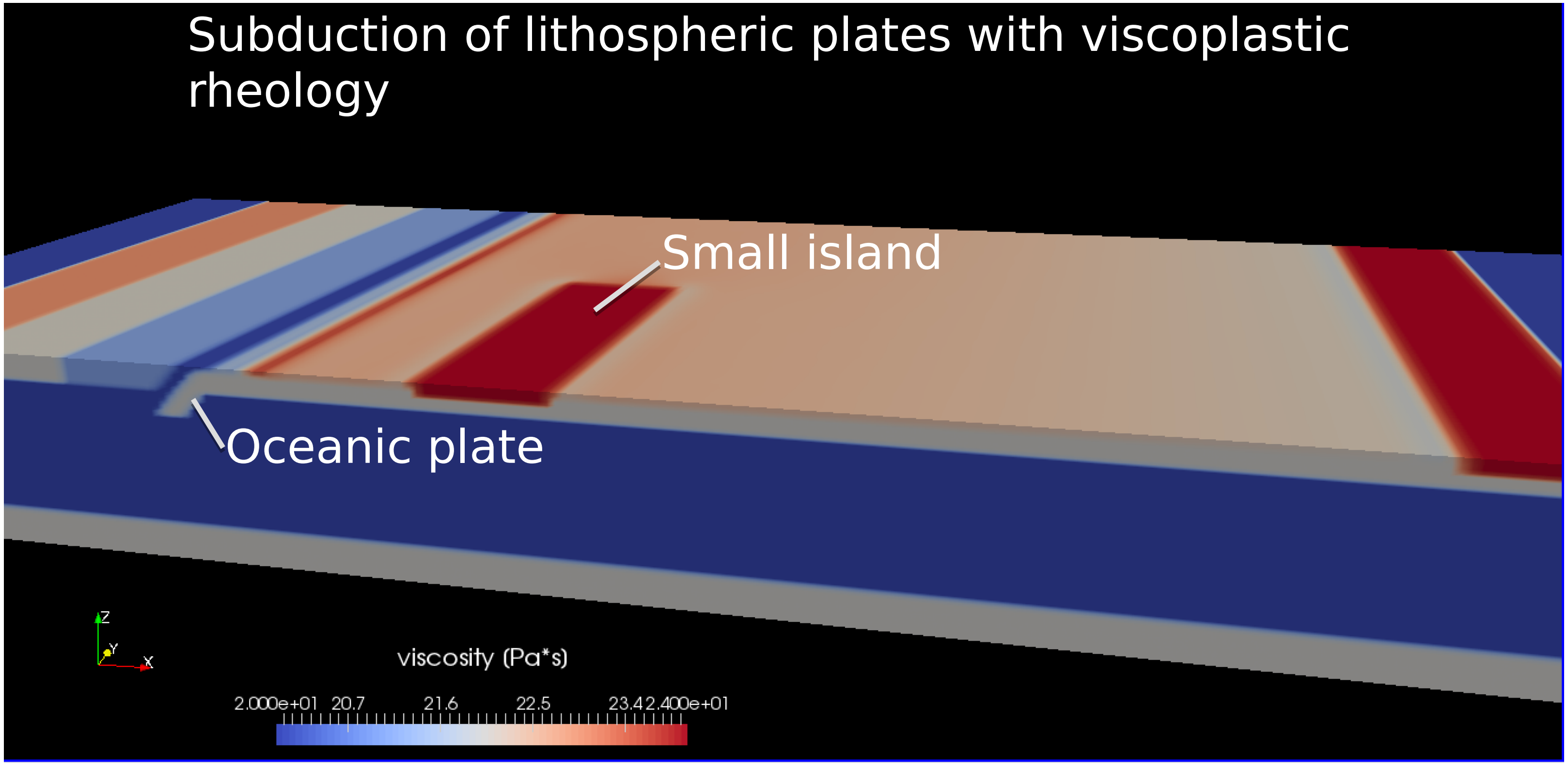
Viscosity sinker=1, viscosity matrix= 1/VC

Multiple spheres is a more tricky problem

Coupled/uncoupled have similar speeds

***Coupled***: 3 GMG levels with FGMRES (rtol 1e-6), Jacobi(20,20) as smoothener; direct coarse grid, 4 cores
***Block***: FGMRES (rtol 1e-6) for full system with 1 V-cycle for the K-block, 3 GMG levels with Jacobi(20,20) as smoother and direct coarse grid

# Coupled vs. Uncoupled MG (typical production run)

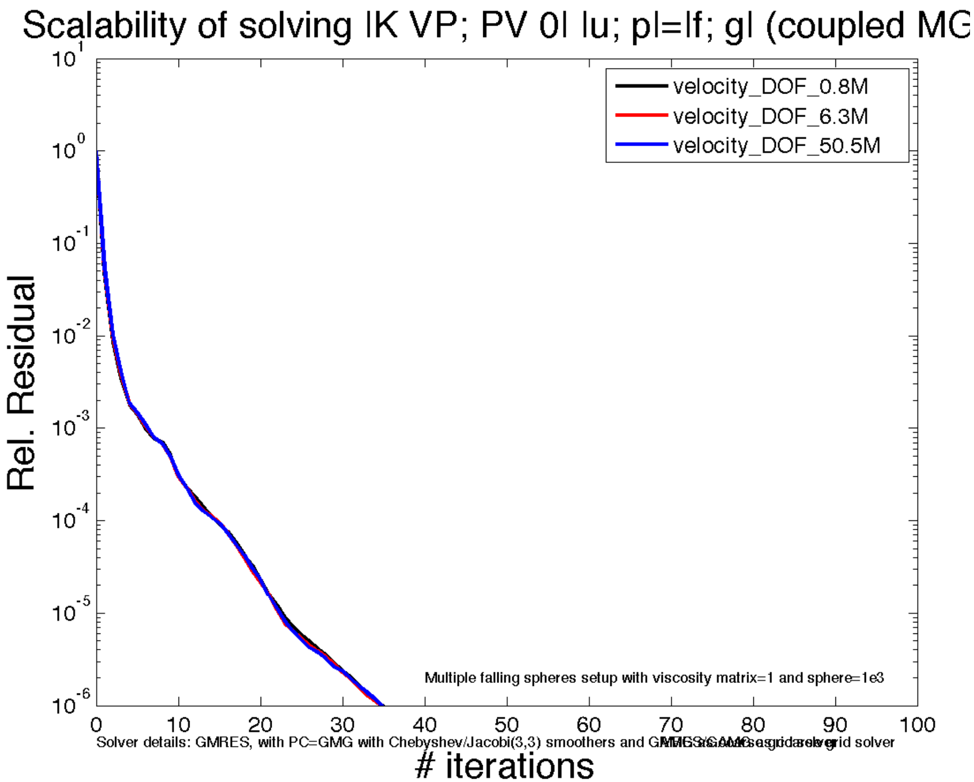


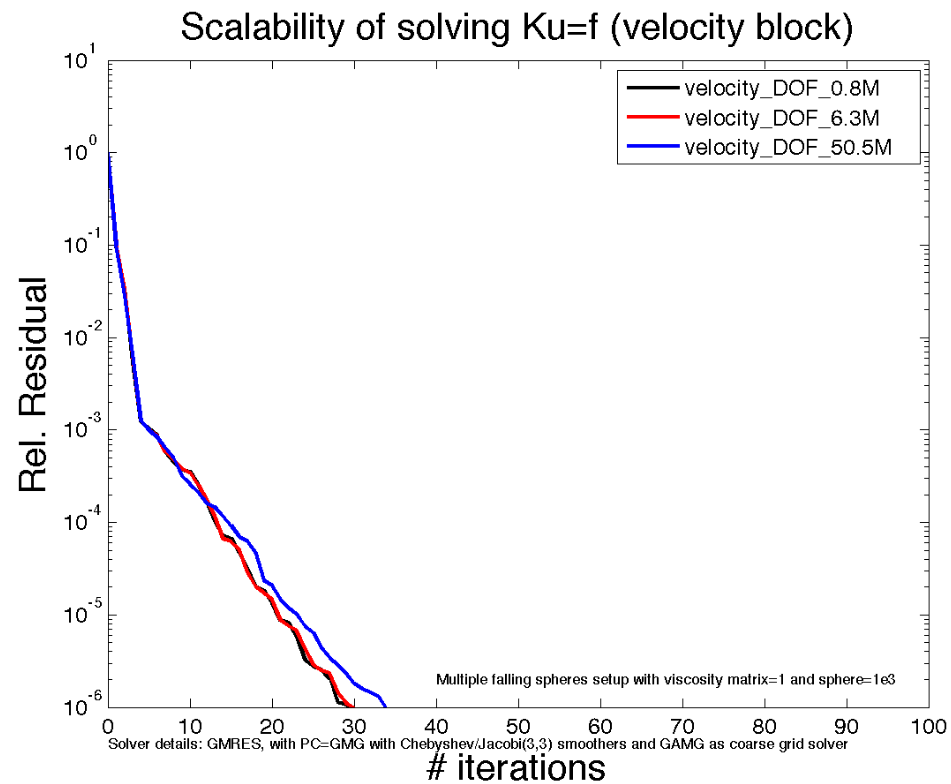

| 3 GMG levels | 128x32x32 | | |
|---|---|---|---|
| **Resolution** | ***# SNES (nonlinear)*** | ***# KSP it (total)*** | ***Time per timestep [s]*** |
| Coupled MG | 2 | 40 | 12 |
| Block MG | 2 | 100 | 24 |

| 4 GMG levels | 256x64x64 | | |
|---|---|---|---|
| **Resolution** | ***# SNES (nonlinear)*** | ***# KSP it*** | ***Time [s]*** |
| Coupled MG | 2 | 30 | 145 |
| Block MG | 5 | 250 | 759 |

Coupled MG is significantly faster for this setup

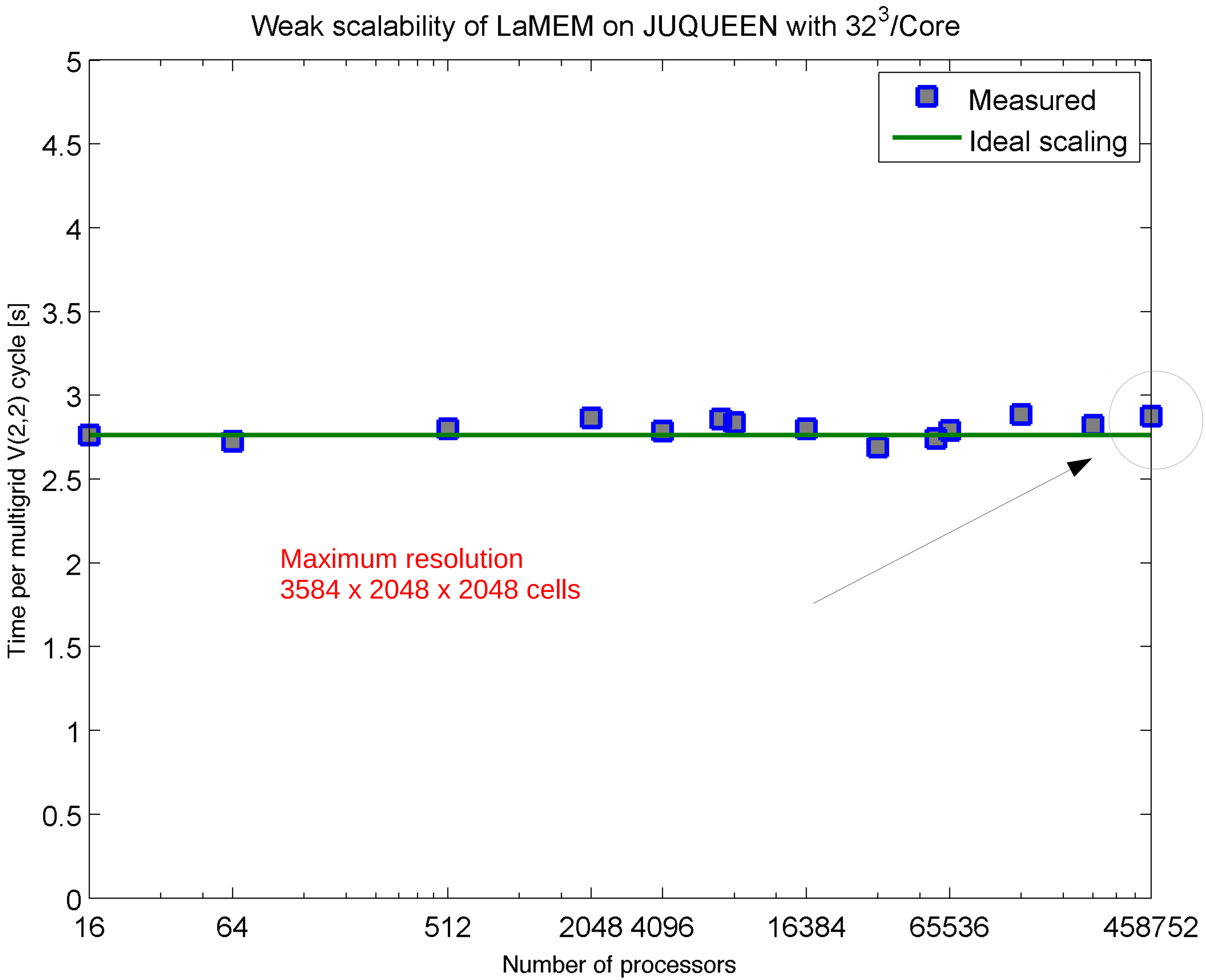
# Weak scaling




| Cores | Total grid size | MG Levels | Coarse Grid size | Velocity DOF | # KSP iterations | Time step [s] | Time/iterations [s] |
|---|---|---|---|---|---|---|---|
| 64 | 128x128x128 | 2 | 64x64x64 | 6.3 Mio | 65 | 155 | 2.38 |
| 512 | 256x256x256 | 3 | 64x64x64 | 50.5 Mio | 67 | 159 | 2.37 |
| 4096 | 512x512x512 | 4 | 64x64x64 | 403 Mio | 71 | 168 | 2.37 |
| 32768 | 1024x1024x1024 | 5 | 64x64x64 | 3.2 Bio | 71 | 209 | 2.94 |
| 65536 | 2048x1024x1024 | 5 | 128x64x64 | 6.4 Bio | 121 | 353 | 2.92 |
| 131072 | 2048x2048x1024 | 5 | 128x128x64 | 12.9 Bio | 112 | 436 | 3.89 |
| 262144 | 2048x2048x2048 | 5 | 128x128x128 | 25.8 Bio | 81 | 482 | 5.95 |

# Weak scaling



# Continuum stress rates

Truesdell, Green-Naghdi and Jaumann rates:

$$\overset{\circ}{\boldsymbol{\sigma}}^{TR} = \dot{\boldsymbol{\sigma}} - \boldsymbol{l}\boldsymbol{\sigma} - \boldsymbol{\sigma}\boldsymbol{l}^{T} + \mathrm{tr}\left[\boldsymbol{l}\right]\boldsymbol{\sigma}$$

$$\overset{\circ}{\boldsymbol{\sigma}}^{GN} = \dot{\boldsymbol{\sigma}} + \boldsymbol{\sigma}\boldsymbol{\Omega} - \boldsymbol{\Omega}\boldsymbol{\sigma}$$

$$\overset{\circ}{\boldsymbol{\sigma}}^{J} = \dot{\boldsymbol{\sigma}} + \boldsymbol{\sigma}\boldsymbol{w} - \boldsymbol{w}\boldsymbol{\sigma}$$

Spatial velocity gradient, spin and angular velocity tensors:

$$\boldsymbol{l} = \dot{\boldsymbol{F}}\boldsymbol{F}^{-1}, \quad \boldsymbol{w} = \frac{1}{2}\left(\boldsymbol{l} - \boldsymbol{l}^{T}\right), \quad \boldsymbol{\Omega} = \dot{\boldsymbol{R}}\boldsymbol{R}^{T}$$

Deformation gradient:

$$\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}$$

Rotation and stretch tensors (polar decomposition)

$$\boldsymbol{F} = \boldsymbol{R}\boldsymbol{U}$$

# Time integrated Jaumann stress rate (2D)

Jaumann rate expresses the rotated stress from previous time step as:

$$\tau_{ij}^{*} = \tau_{ij}^{n} + \Delta t \left( w_{ik}\tau_{kj}^{n} - \tau_{ik}^{n} w_{kj} \right)$$

which can be viewed (e.g. Beuchert and Podladchikov, 2010) as a truncated Taylor series expansion of a simple stress rotation formula:

$$\tau_{ij}^{*} = R_{ik}\tau_{kl}^{n} R_{jl}$$

Jaumann rate expression imposes severe time step restrictions.
It is common (e.g. Geria, 2010) to use the original stress rotation formula and estimate rotation angle from the time integration of the vorticity field:

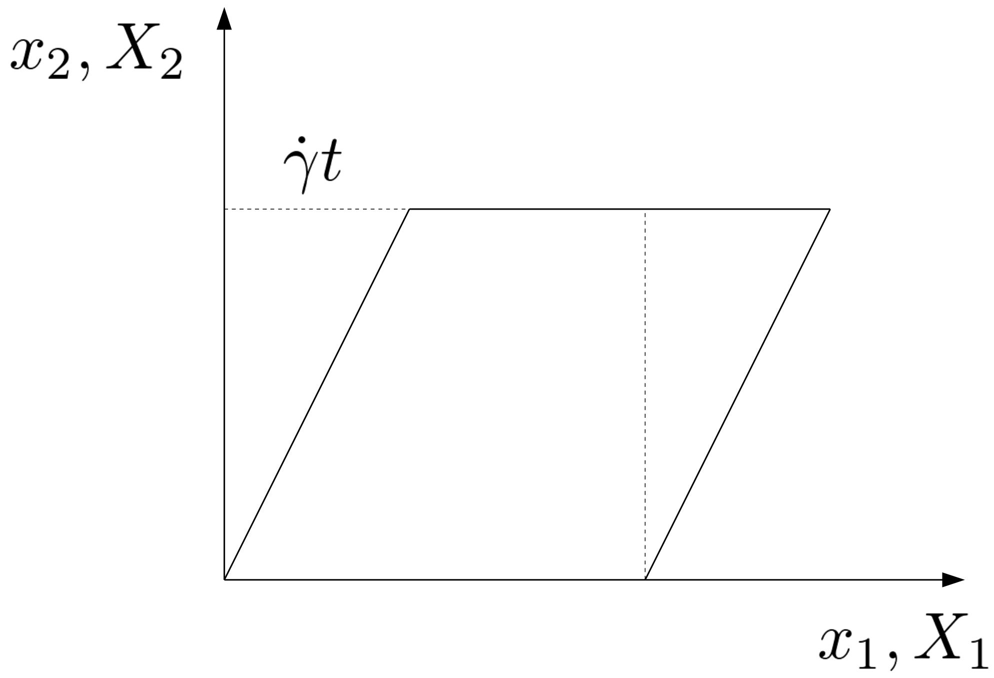
$$\theta = \frac{1}{2}\left( \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \Delta t$$

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Beuchert and Podladchikov (2010). Viscoelastic mantle convection and lithospheric stresses. Geophys. J. Int., 183, 35–63.

Gerya (2010) Introduction to numerical Geodynamic Modelling.

# Simple shear test




Material motion:

$$
\begin{aligned}
x_1 &= X_1 + \dot\gamma t X_2 \\
x_2 &= X_2
\end{aligned}
$$

Deformation gradient:

$$
\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = \begin{pmatrix} 1 & \dot\gamma t \\ 0 & 1 \end{pmatrix}
$$

Spatial velocity gradient, rate of deformation, and spin tensors:

$$
\boldsymbol{l} = \dot{\boldsymbol{F}}\boldsymbol{F}^{-1} = \begin{pmatrix} 0 & \dot\gamma \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{d} = \begin{pmatrix} 0 & \frac{\dot\gamma}{2} \\ \frac{\dot\gamma}{2} & 0 \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} 0 & \frac{\dot\gamma}{2} \\ -\frac{\dot\gamma}{2} & 0 \end{pmatrix}
$$

Rotation tensor (polar decomposition) and angular velocity tensor:

$$
\boldsymbol{R} = \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix}, \quad \tan\beta = \frac{\dot\gamma t}{2}, \quad \boldsymbol{\Omega} = \dot{\boldsymbol{R}}\boldsymbol{R}^T = \left[1 + \left(\frac{\dot\gamma t}{2}\right)^2\right]^{-1} \boldsymbol{w}
$$

# Elastic case

Neo-Hookean (elastic stored energy function):

$$W = \frac{G}{2}\left(\overline{I}_1 - 3\right) + \frac{K}{2}\left(J - 1\right)^2$$

Determinant of deformation gradient (volume change):

$$J = \det\left[\boldsymbol{F}\right]$$

Volume-preserving left Cauchy-Green deformation tensor & first invariant:

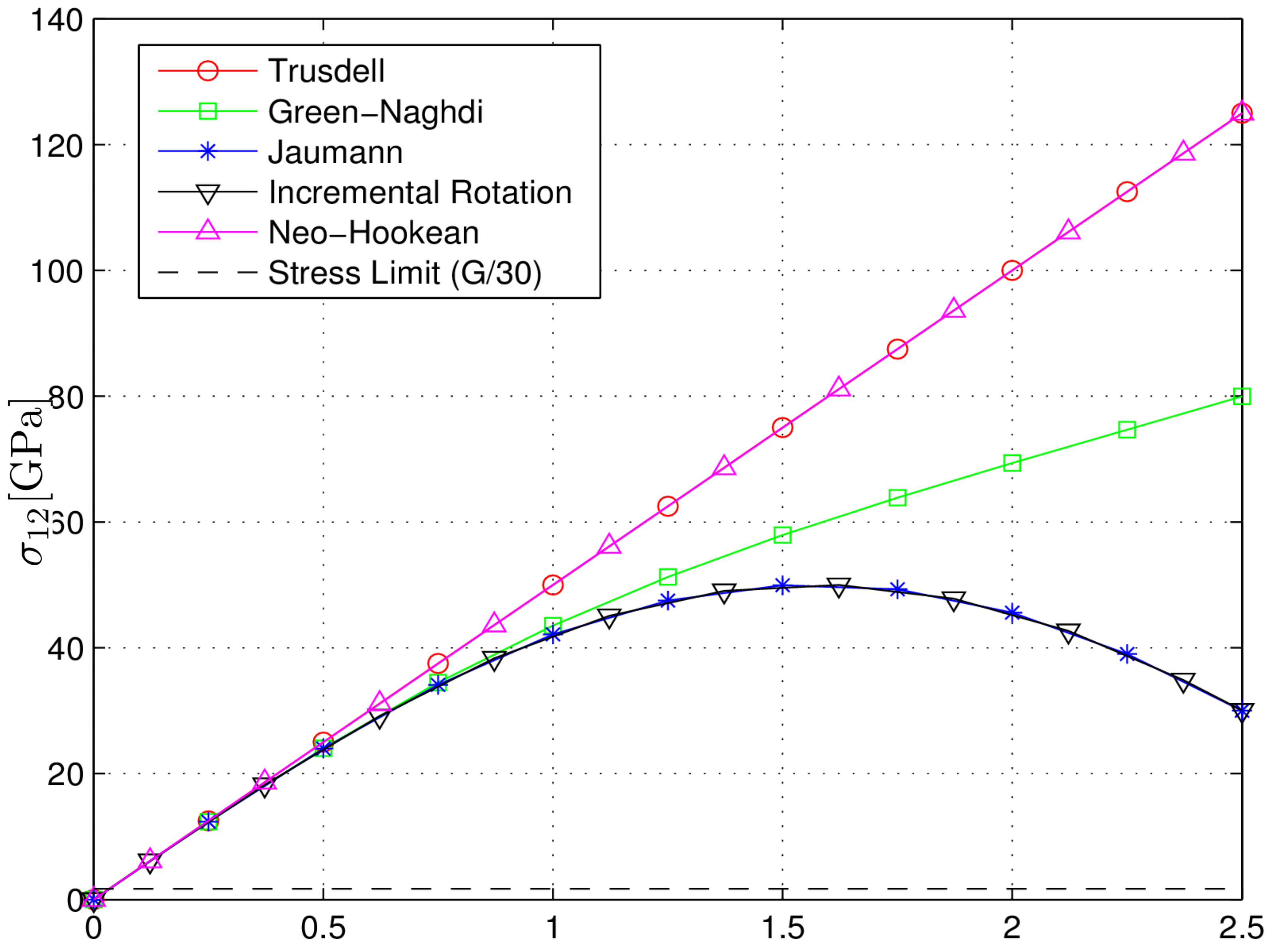
$$\overline{\boldsymbol{b}} = J^{-2/3}\boldsymbol{F}\boldsymbol{F}^T, \quad \overline{I}_1 = \operatorname{tr}\left[\,\overline{\boldsymbol{b}}\,\right]$$

Cauchy stress:

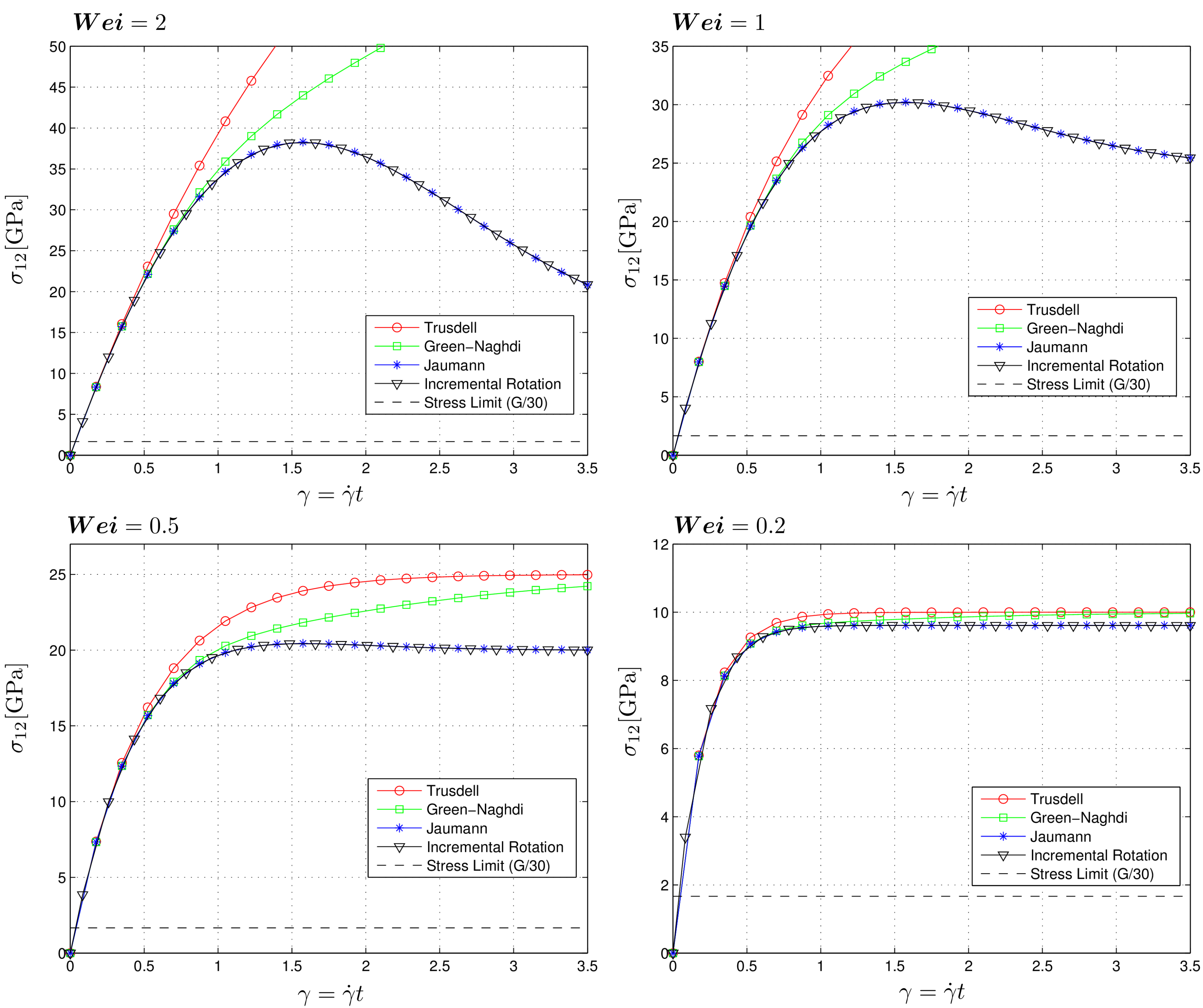
$$\boldsymbol{\sigma} = J^{-1}\boldsymbol{F}\frac{\partial W}{\boldsymbol{F}} = J^{-1}G \operatorname{dev}\left[\,\overline{\boldsymbol{b}}\,\right] + K(J-1)\mathbf{1}$$

$$\boldsymbol{\sigma} = G \operatorname{dev}\left[\boldsymbol{b}\right] = G\begin{pmatrix} \frac{2}{3}(\dot{\gamma}t)^2 & \dot{\gamma}t & 0 \\ \dot{\gamma}t & -\frac{1}{3}(\dot{\gamma}t)^2 & 0 \\ 0 & 0 & -\frac{1}{3}(\dot{\gamma}t)^2 \end{pmatrix}$$

# Elastic case



# Visco-Elastic case (Weissenberg number)

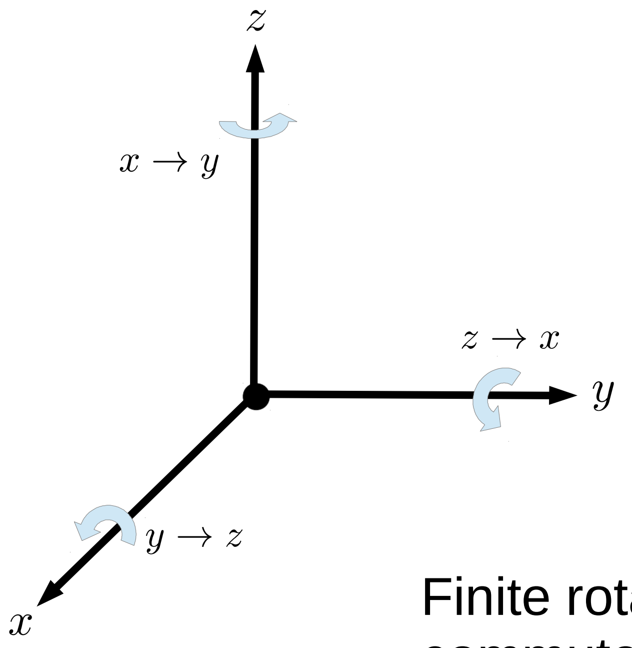


# 3D generalization of 2D incremental rotation rate

Can we use a similar method in 3D?

In 2D vorticity pseudo-vector has a single component and instantaneous rotation axis is always perpendicular to the plane.

In 3D vorticity pseudo-vector has three components and instantaneous rotation axis can change in time:




$$\begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} \frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \\ \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \\ \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \end{pmatrix}$$

Positive rotation directions: Counter-clockwise
Coordinate system: Right-handed

Finite rotations around coordinate axis unfortunately do not commute in 3D. Nevertheless the 3D generalization of a 2D algorithm yields reasonable results (Rubinstein and Atluri, 1983).

Rubinstein and Atluri (1983). Objectivity of incremental constitutive relations over finite time steps in computational finite deformation analyses. Comput. Methods Appl. Mech. Engrg., 36, 277-290

# 3D generalization of 2D incremental rotation rate

The 3D algorithm can be summarized as follows:

[1] Compute vorticity vector magnitude:

$$w = \sqrt{w_x^2 + w_y^2 + w_z^2}$$

[2] Compute unit rotation axis:

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = w^{-1} \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix}$$

[3] Integrate incremental rotation angle:

*(average angular velocity is two times smaller than the vorticity vector magnitude)*

$$\theta = \Delta t \left( \frac{w}{2} \right)$$

[4] Evaluate rotation matrix using Euler-Rodrigues formula:

$$R = \cos\theta \begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} + \sin\theta \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix} + (1 - \cos\theta) \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_y n_x & n_y^2 & n_y n_z \\ n_z n_x & n_z n_y & n_z^2 \end{pmatrix}$$

[5] Rotate stress: $\tau^* = R\,\tau^n R^T$

# Comparison with Jaumann stress rate

Consider a rather extreme case of 3D accelerated periodic rotational motion:

$$v_x = r\cos\theta\dot{\theta}\cos\phi - r\sin\theta\sin\phi\dot{\phi}$$

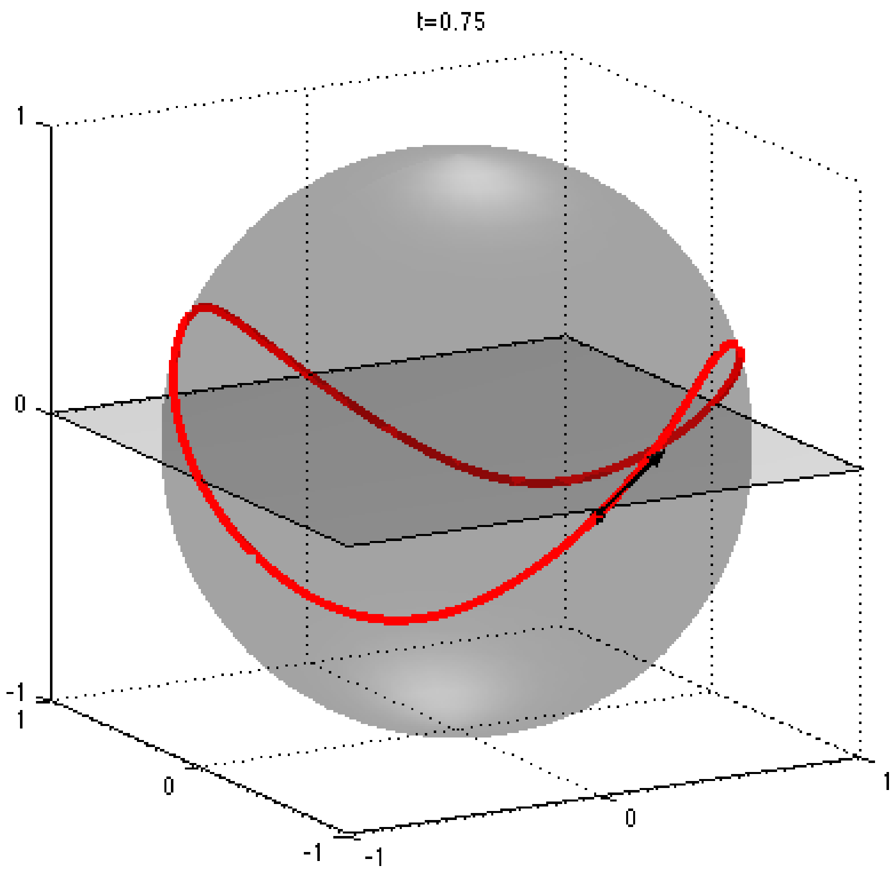
$$v_y = r\cos\theta\dot{\theta}\sin\phi + r\sin\theta\sin\phi\dot{\phi}$$

$$v_z = -r\sin\theta\dot{\theta}$$

$$\phi(t) = 2\pi t^5$$

$$\theta(t) = \frac{\pi}{2} + \frac{\pi}{8}\sin(4\pi t^5)$$

Point trajectory




Assign randomly the stress tensor components at the initial time and integrate by thee different methods:

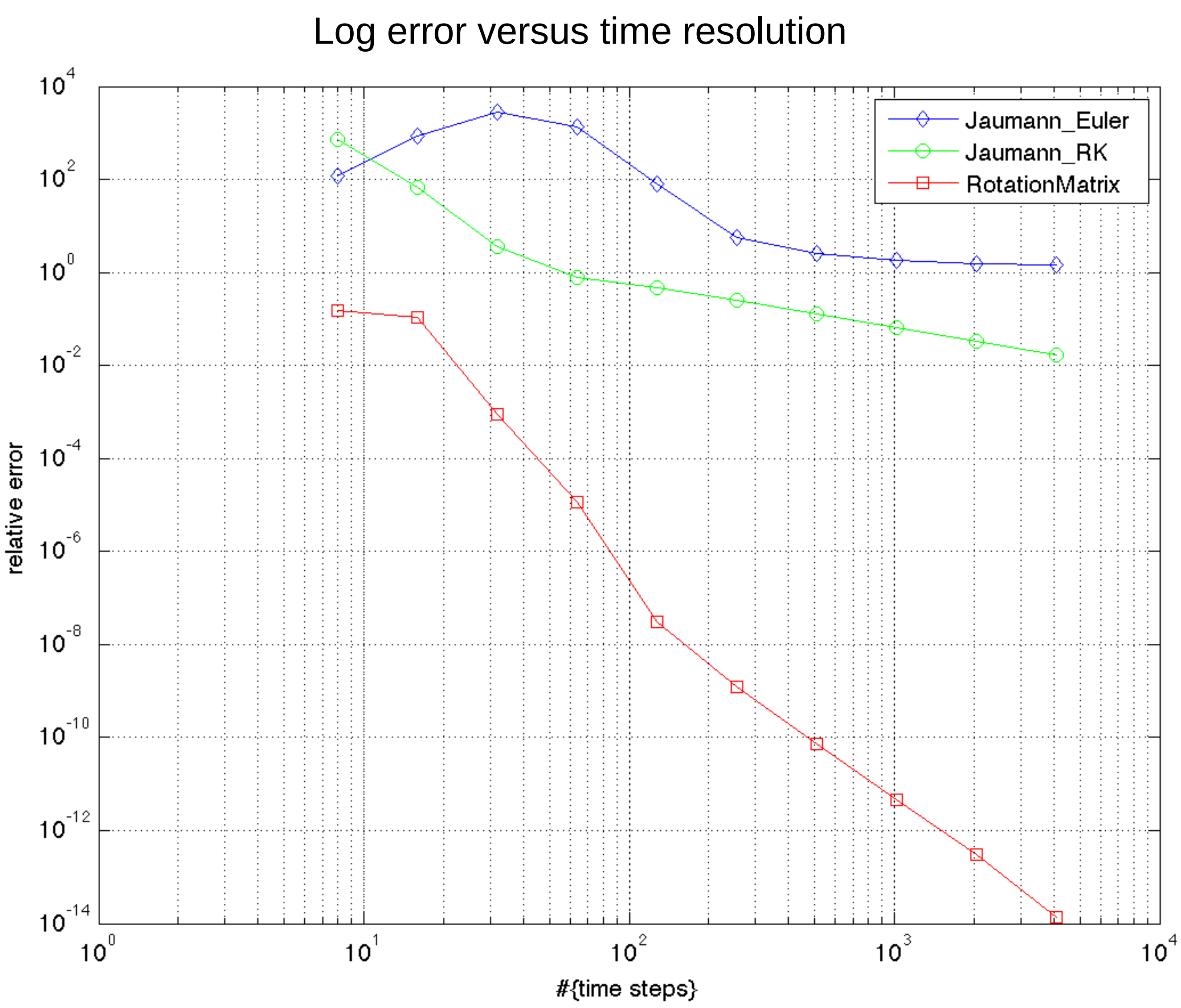
[A] Jaumann forward Euler

[B] Jaumann 4-th order Runge-Kutta

[C] Generalized 3D rotation matrix

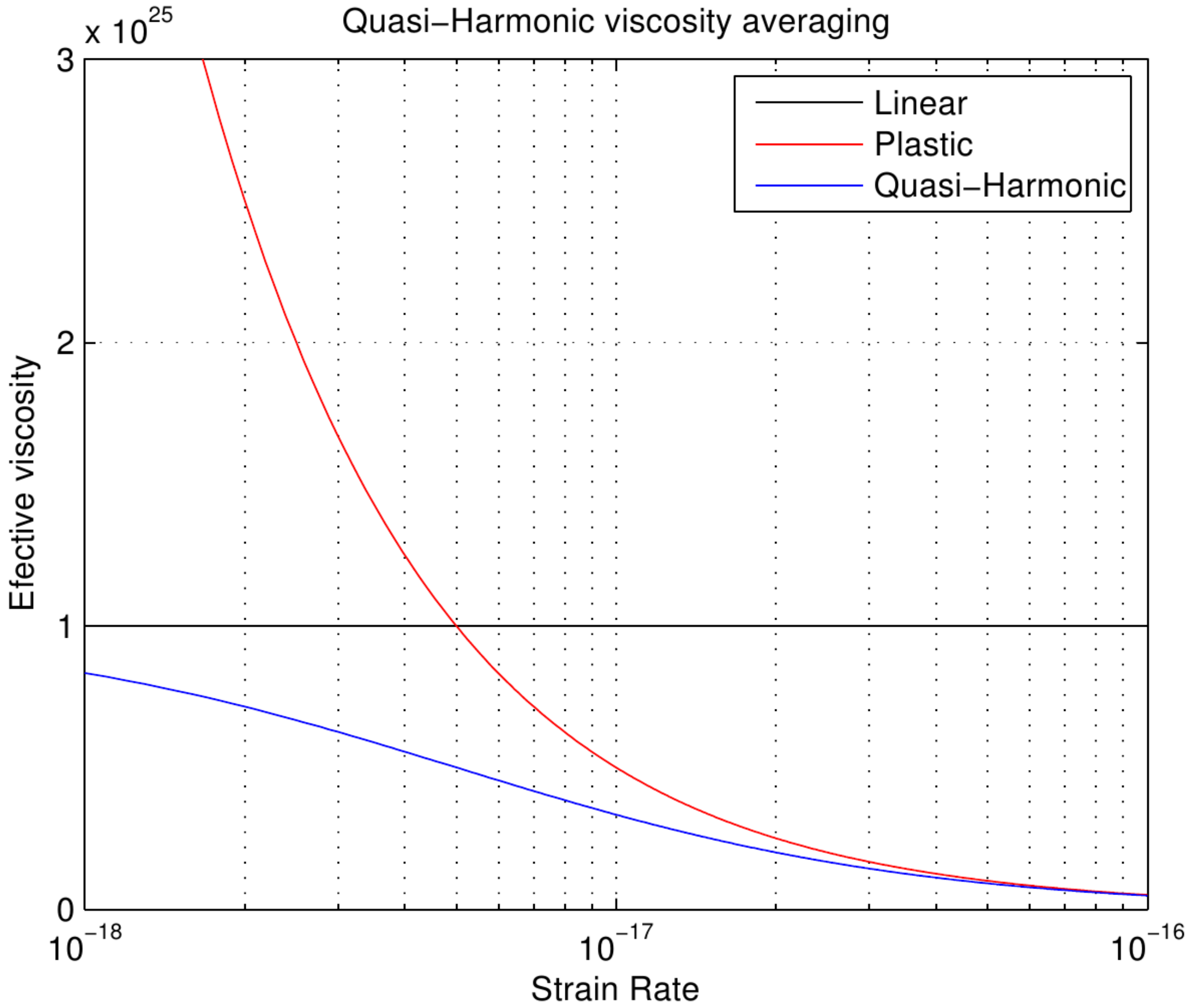
After a full turn the stress components should the same as in the beginning.

# Comparison with Jaumann stress rate




Conclusion: 3D rotation matrix yields much better results than Jaumann rate

# Minimum vs. quasi-harmonic plastic viscosity




Plastic viscosity:

$$\eta_p = \frac{\tau_Y}{2\dot{\varepsilon}^*_{II}}$$

Minimum viscosity:

$$\eta^* = \min(\eta, \eta_p)$$

Quasi-harmonic viscosity:

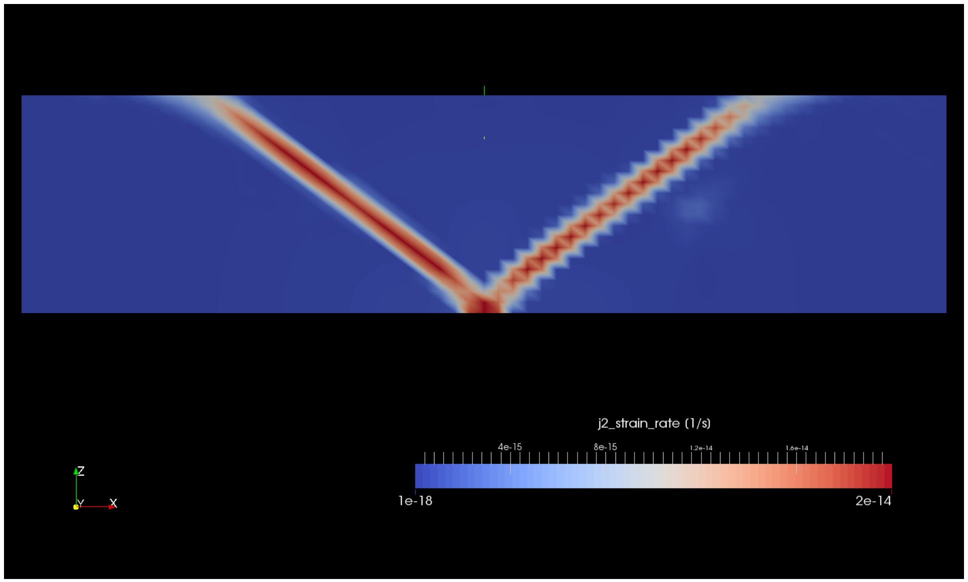
$$\eta^* = \left(\frac{1}{\eta}, \frac{1}{\eta_p}\right)^{-1}$$

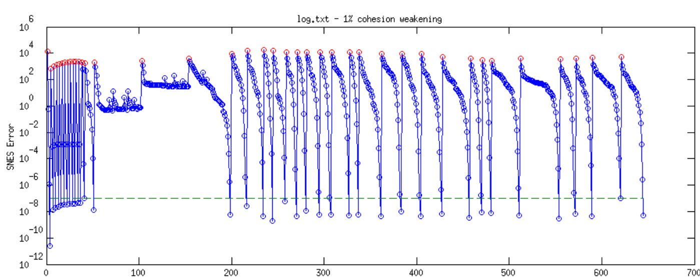
Both are coincident only at high strain rates

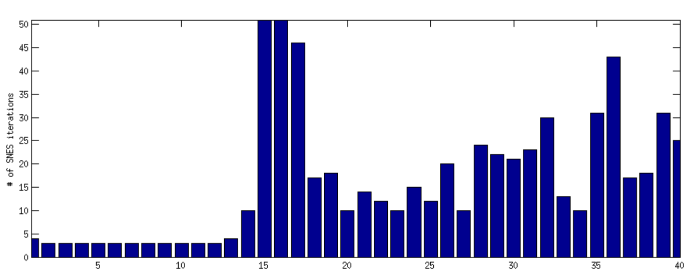
Quasi-harmonic has spurious plastic deformation below yield!

# Minimum vs. quasi-harmonic plastic viscosity

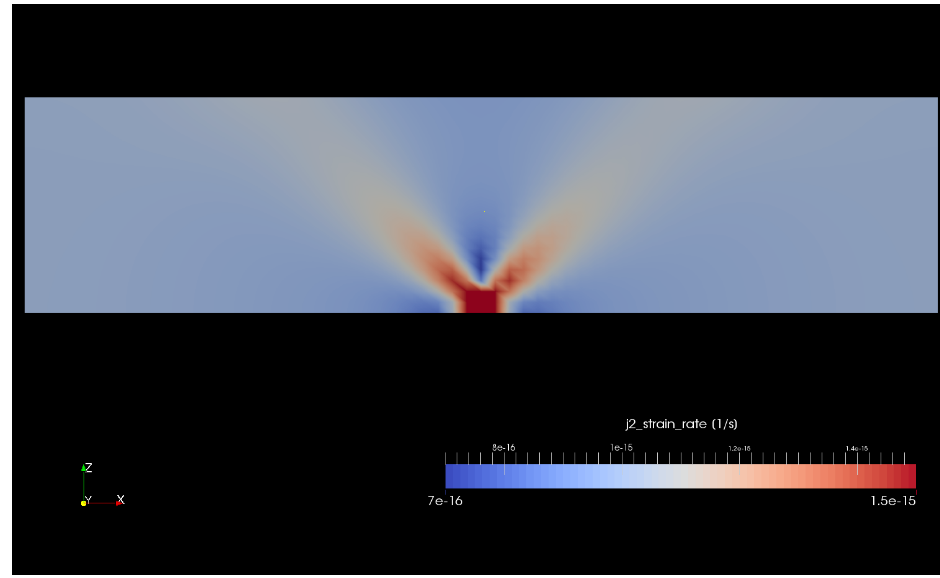
Minimum viscosity model
Hard to converge, sharp localization

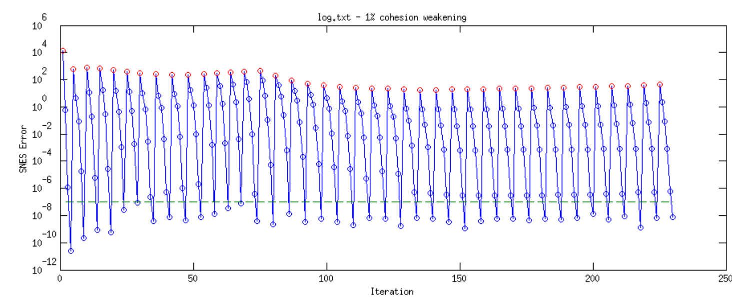


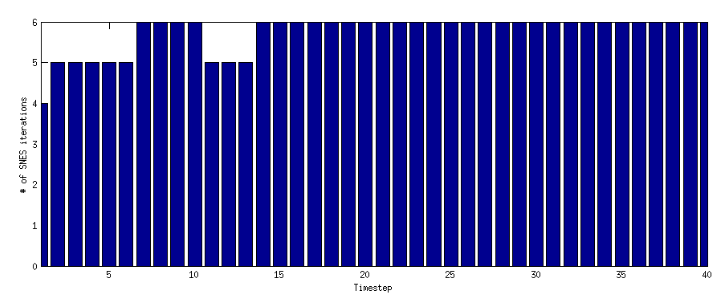



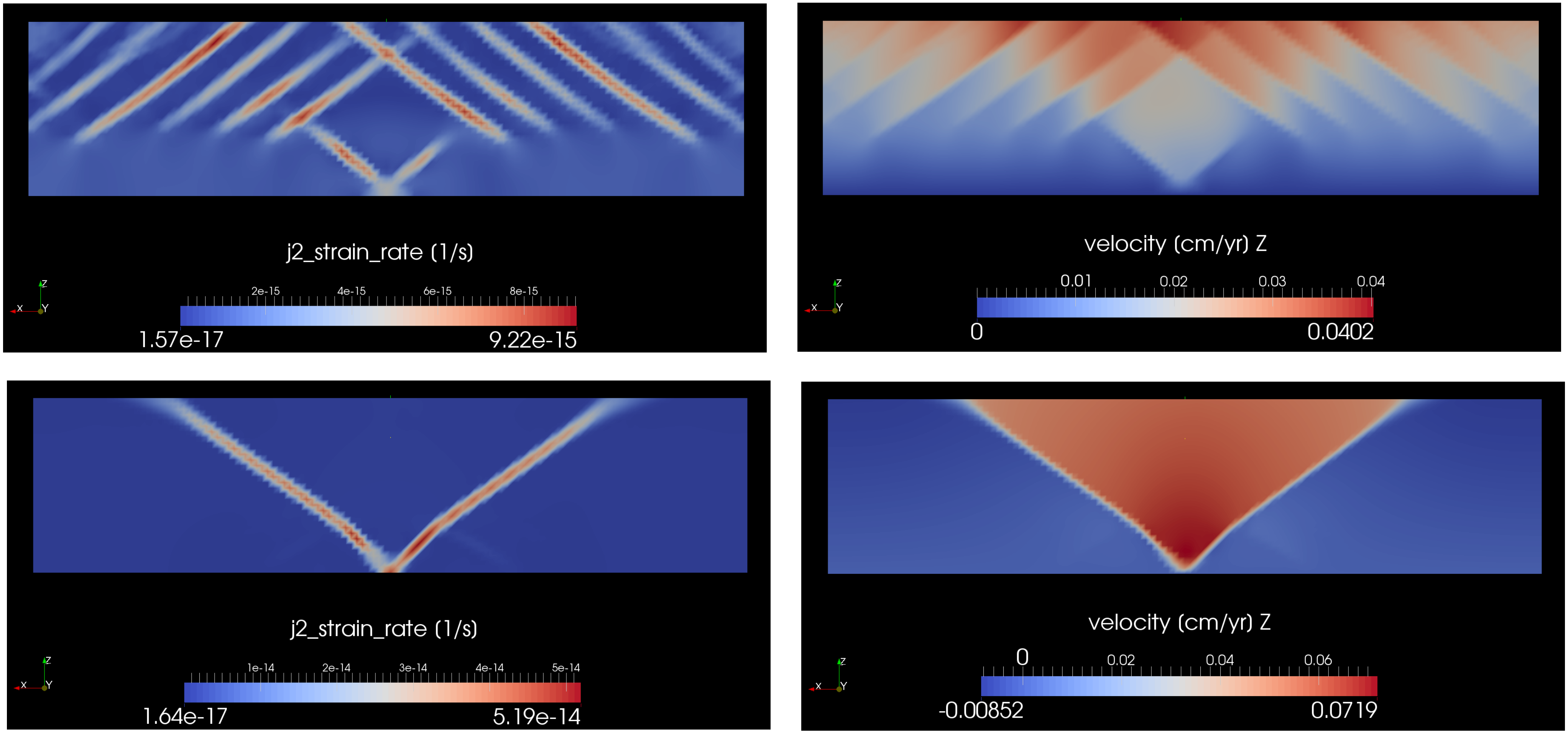

Quasi-harmonic viscosity model
Easy to converge, very pure localization



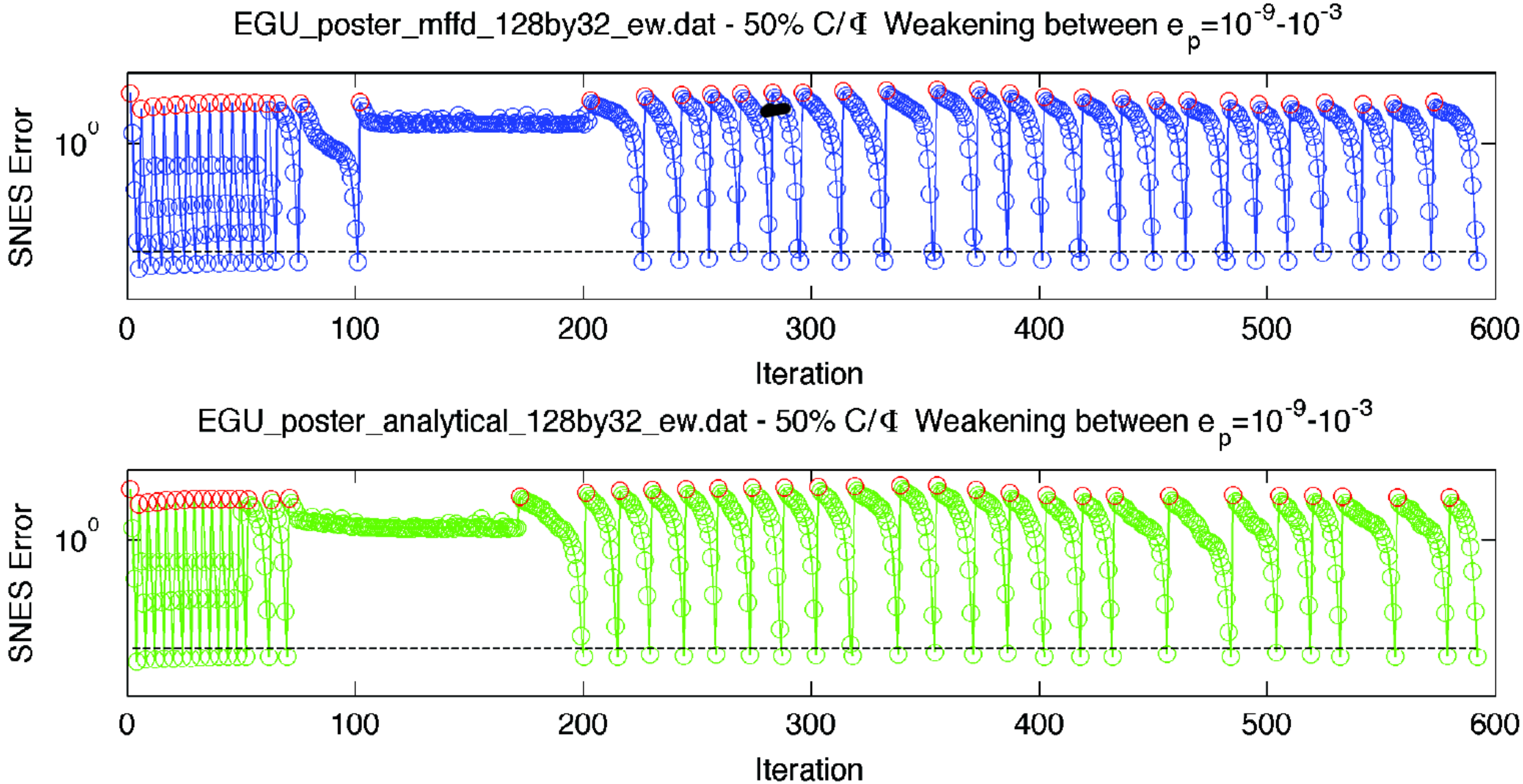
# Plasticity convergence issues

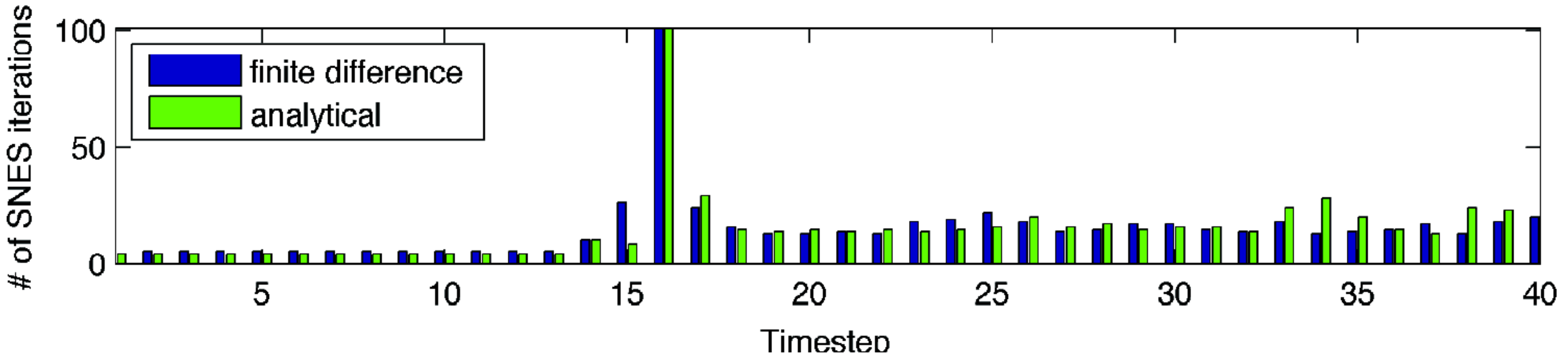



Plastic localization setup

Drucker-Prager elasto-plastic rheology

# Plasticity convergence issues

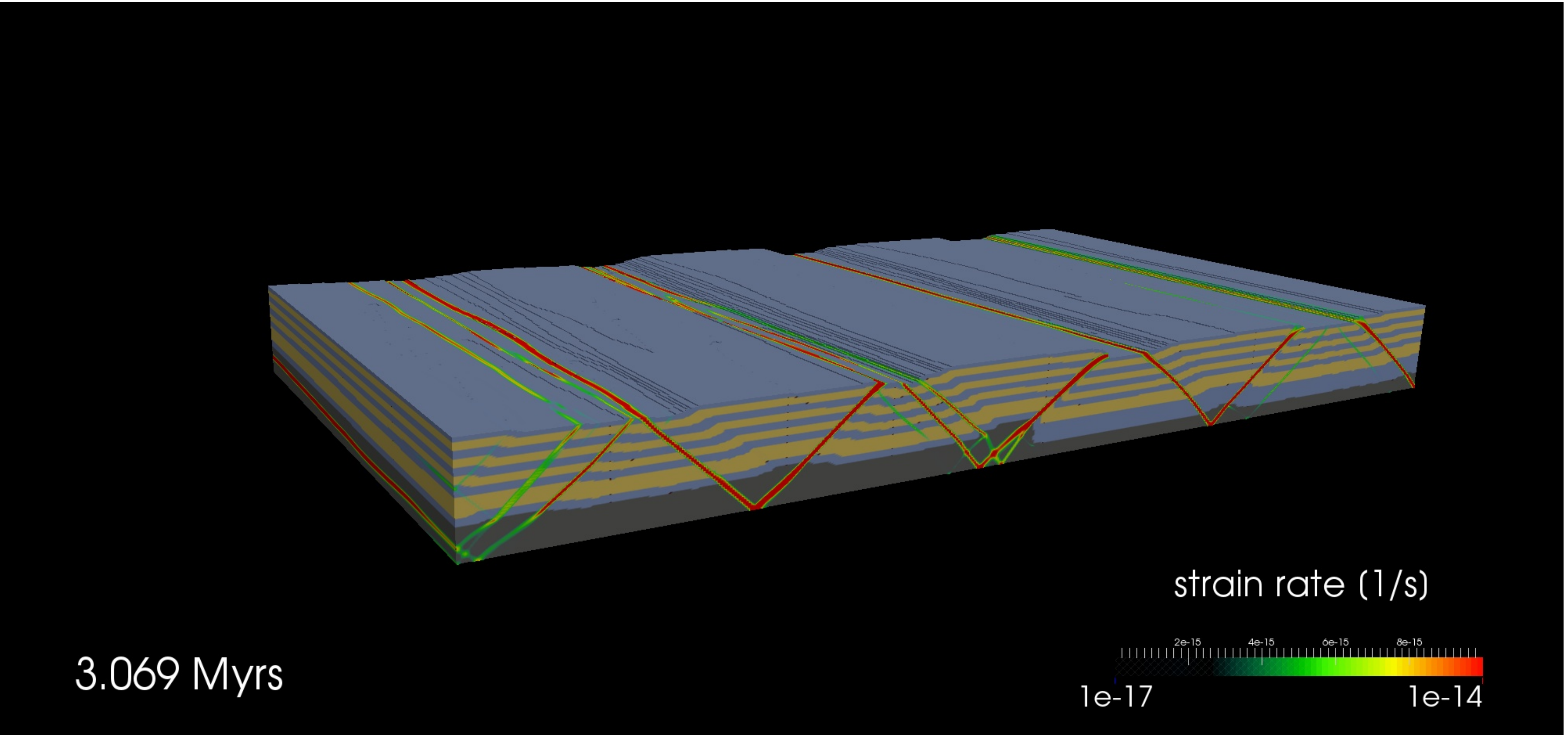


# Plasticity convergence issues (summary)

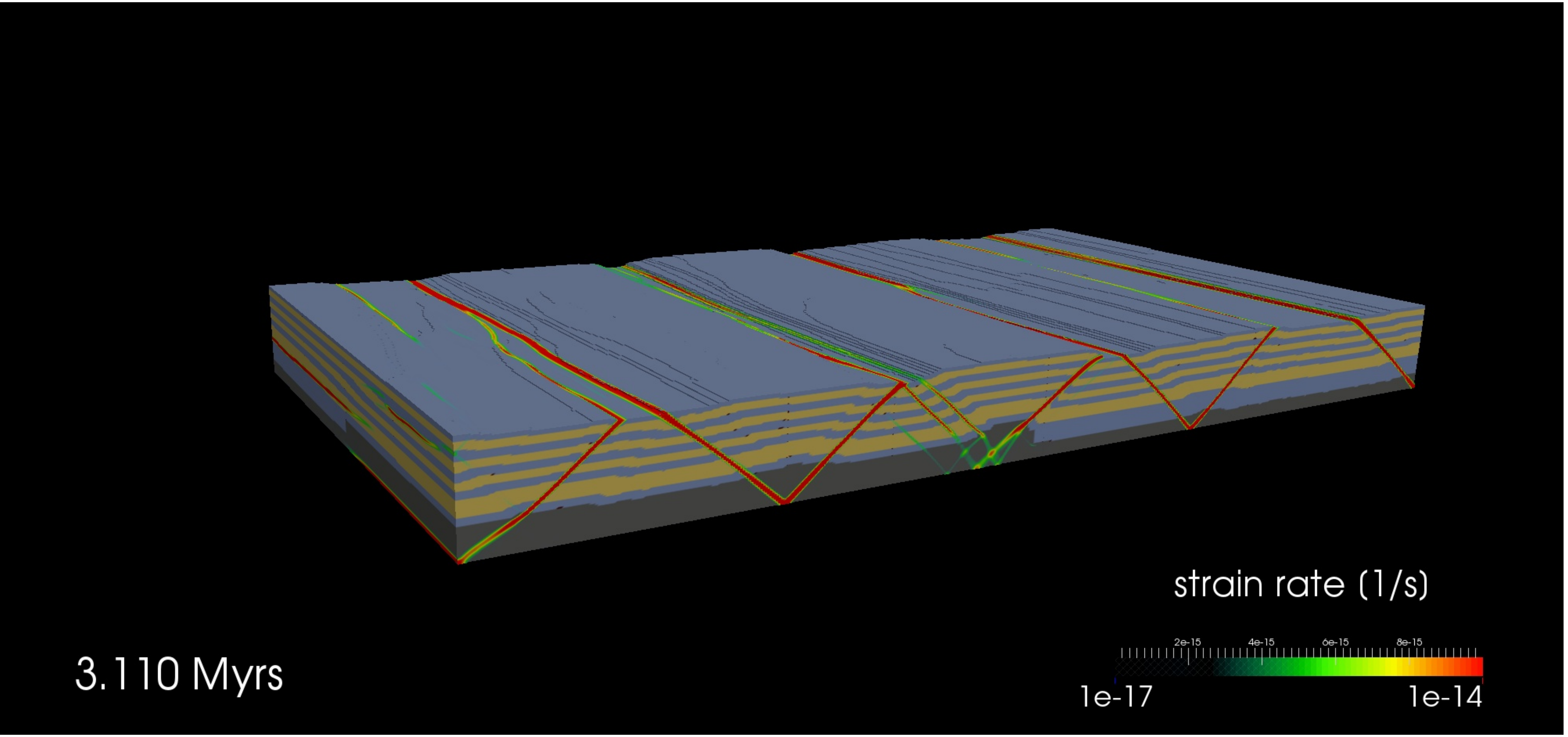
- Elasto-plastic setups converge better than visco-plastic
- Strain softening facilitates convergence
- Sometimes non-convergent solutions are reasonable (continuation is possible)
- A combination of Newton and Picard is necessary
- Line search and Eisenstat-Walker algorithms are helpful
- Visco-plastic pressure-dependent rheology may be not universally solvable (Spiegelman et al., 2016) despite quasi-harminic averaging.
- Quasi-harmonic averaging produces pure localization, but fast to converge
- Analytical Jacobian doesn't help to accelerate convergence

# 3D Multilayer detachment folding (2D heterogeneity)



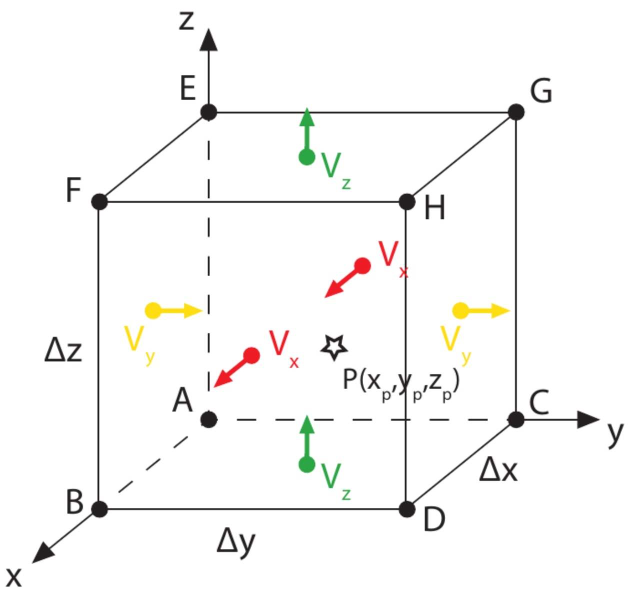

Grid resolution: 512 x 256 x 128 cells

# 3D Multilayer detachment folding (3D heterogeneity)




Grid resolution: 512 x 256 x 128 cells

# Conservative velocity interpolation (CVI)




Wang et al. [2015]

Enough for FE

Not enough for FDSTAG

Velocities are not in the corners

Prevent unphysical marker dispersion:

$$V_i^P = V_i^L + \Delta V_i$$

Liner interpolation:

$$\begin{aligned} V_i^L(x_p, y_p, z_p) = &(1-x_p) \times (1-y_p) \times [(1-z_p) \times V_i^A + z_p \times V_i^E] + \\ &x_p \times (1-y_p) \times [(1-z_p) \times V_i^B + z_p \times V_i^F] + \\ &(1-x_p) \times y_p \times [(1-z_p) \times V_i^C + z_p \times V_i^G] + \\ &x_p \times y_p \times [(1-z_p) \times V_i^D + z_p \times V_i^H], \end{aligned}$$

Correction:

$$\Delta V_x = x_p(1-x_p)(C_{10} + z_p C_{12})$$

$$\Delta V_y = y_p(1-y_p)(C_{30} + x_p C_{31})$$

$$\Delta V_z = z_p(1-z_p)(C_{20} + y_p C_{23})$$

$$C_{12} = \frac{\Delta x}{2\Delta y}[-V_y^A + V_y^B + V_y^C - V_y^D + V_y^E - V_y^F - V_y^G + V_y^H]$$

$$C_{23} = \frac{\Delta z}{2\Delta x}[-V_x^A + V_x^B + V_x^C - V_x^D + V_x^E - V_x^F - V_x^G + V_x^H]$$

$$C_{31} = \frac{\Delta y}{2\Delta z}[-V_z^A + V_z^B + V_z^C - V_z^D + V_z^E - V_z^F - V_z^G + V_z^H]$$

$$C_{10} = \frac{\Delta x}{2\Delta z}[V_z^A - V_z^B - V_z^E + V_z^F] + \frac{\Delta x}{2\Delta y}[V_y^A - V_y^B - V_y^C + V_y^D + C_{31}]$$

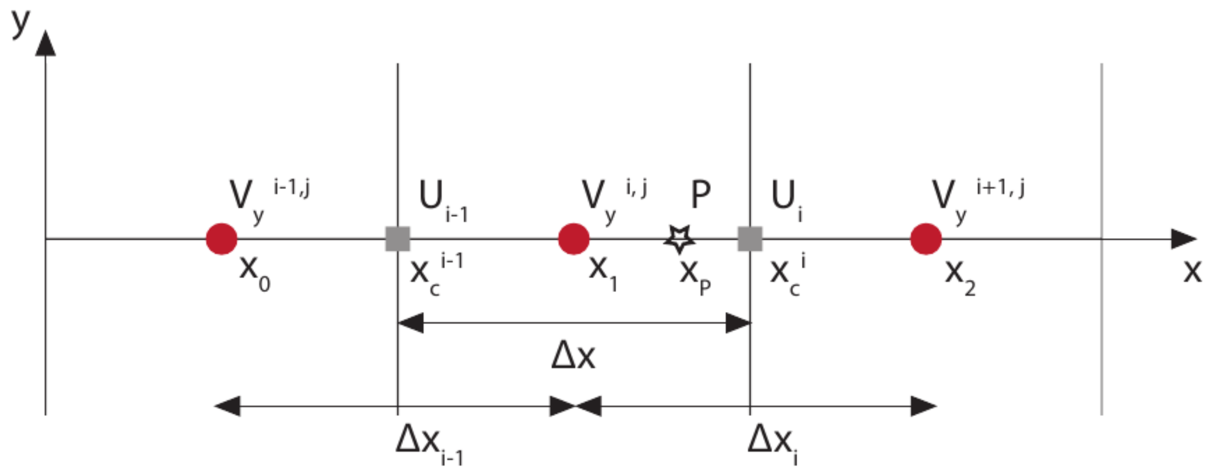
$$C_{20} = \frac{\Delta z}{2\Delta x}[V_x^A - V_x^B - V_x^E + V_x^F + C_{12}] + \frac{\Delta z}{2\Delta y}[V_y^A - V_y^C - V_y^E + V_y^G]$$

$$C_{30} = \frac{\Delta y}{2\Delta x}[V_x^A - V_x^B - V_x^C + V_x^D] + \frac{\Delta y}{2\Delta z}[V_z^A - V_z^C - V_z^E + V_z^G + C_{23}]$$

# Minmod Interpolant

Interpolate velocities from faces to corners:

- linear - Lin
- quadratic - Q2
- spline quadratic SQ2
- Minmod

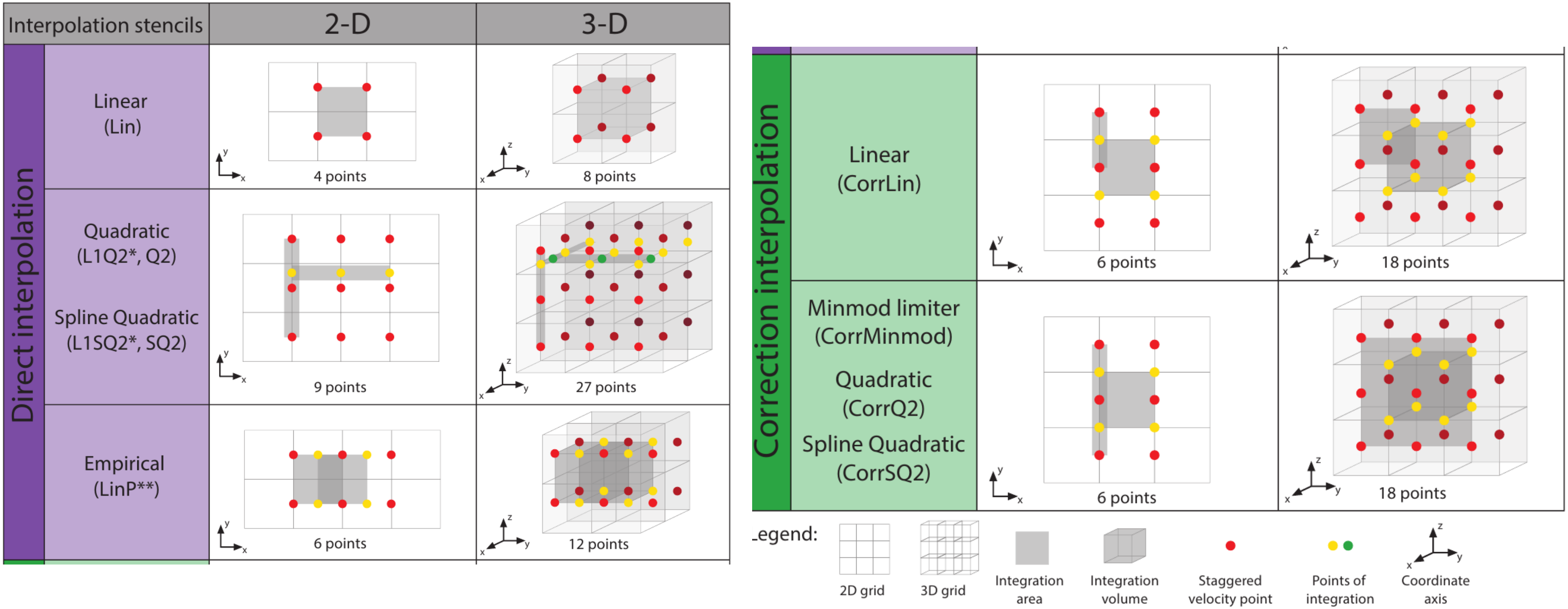



$$U_i = V_y^{i,j} - \frac{\Delta x}{2} minmod \left( \frac{V_y^{i+1,j} - V_y^{i,j}}{\Delta x_i}, \frac{V_y^{i,j} - V_y^{i-1,j}}{\Delta x_{i-1}} \right)$$

$$minmod(A,B) = \begin{cases} A, & \text{if } A \cdot B > 0 \text{ and } |A| \leqslant |B| \\ B, & \text{if } A \cdot B > 0 \text{ and } |A| > |B| \\ 0, & \text{if } A \cdot B \leqslant 0 \end{cases}$$

Jenny et al. [2001] and Meyer and Jenny [2004]

# Conservative velocity interpolation (CVI)




Püsök et al. [2016] (Submitted)

Direct interpolation methods dont' apply correction

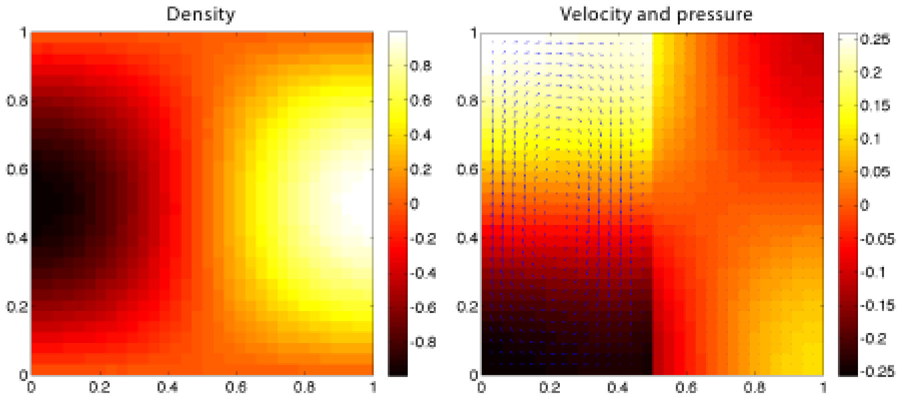
**LinP** interpolates from corners and pressure nodes cell centers (T. Gerya, private communication)

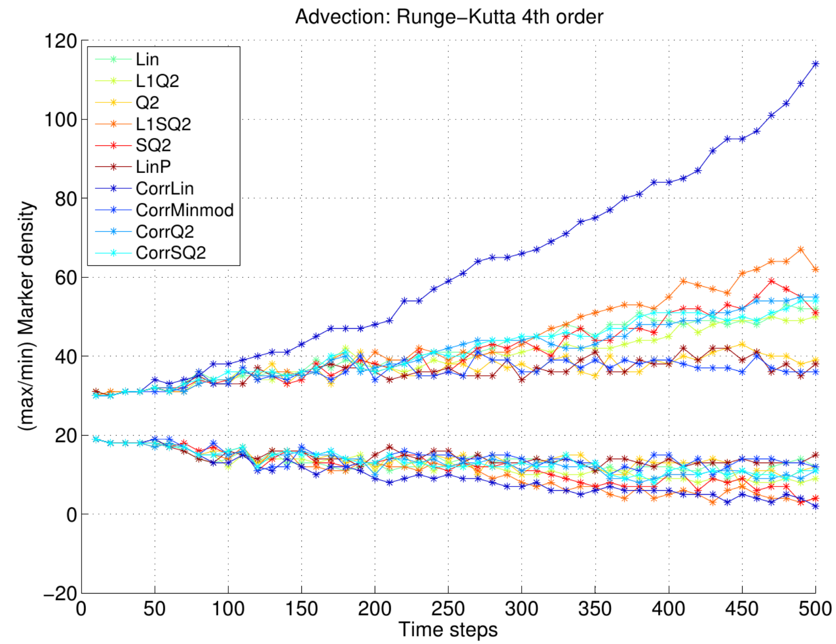
$$V_i^P = AV_i^{Lin} + (1-A)V_i^{pressure}$$

Corrected interpolation methods apply correction after different interpolation to the corners

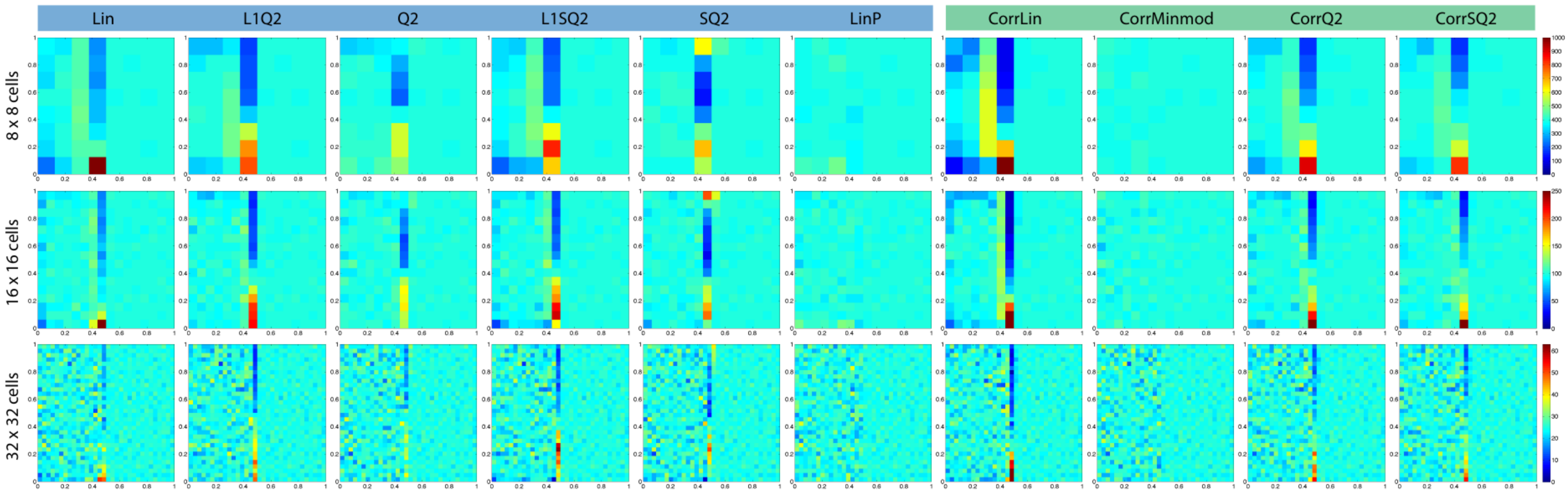
# Velocity interpolation comparison

SolCx benchmark




Marker density distribution




LinP (Gerya) and CorrMinmod produce best results.
More details are coming soon Püsök et al. [2016]

# Gradient-based inversion methods

Minimize misfit function (F), formulated in terms of model parameters (p)

$$\min_p (F(p))$$

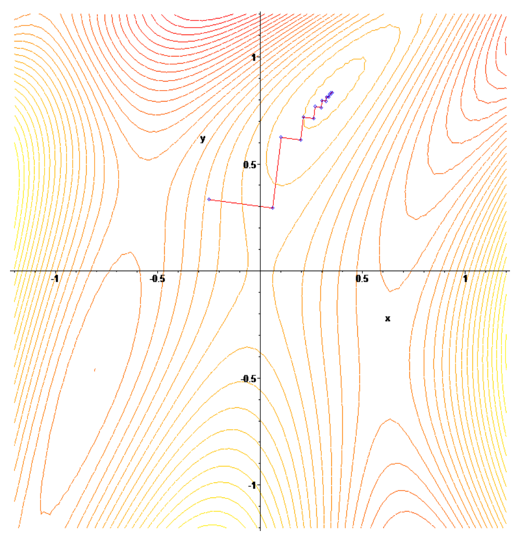
Gradient descent: $p_{n+1} = p_n - \gamma\, G(p_n)$ $\qquad G = \dfrac{dF}{dp}$ - gradient vector

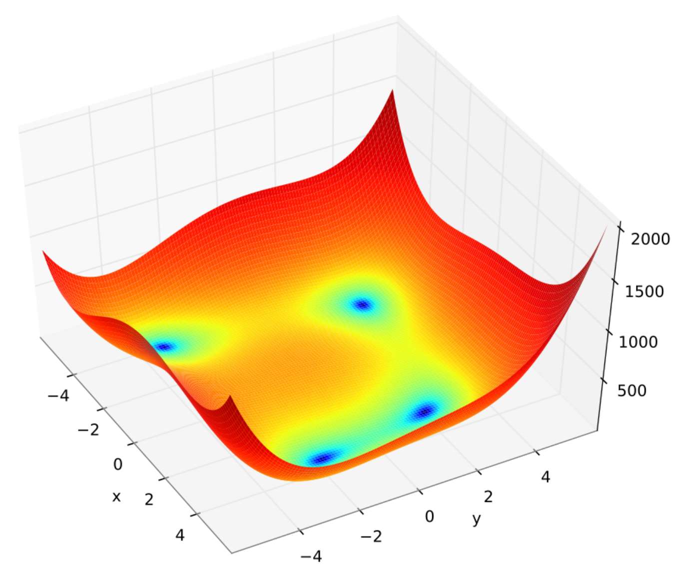
Newton method: $p_{n+1} = p_n - \alpha\, H(p_n)^{-1} G(p_n)$ $\qquad H = \dfrac{dG}{dp}$ - Hessian matrix

Hessian matrix can be efficiently approximated by BFGS algorithm

DISADVANTAGES:

- *Requires derivatives*
- *Doesn't sample misfit function*
- *Sensitive to local minima*
- *Unstable slow convergence*






ADVANTAGES:

- *Relatively simple*
- *Works for many parameters*

# Efficient adjoint gradient evaluation

Misfit function (F) is normally defined using the forward problem solution (x):

$$F(x, x(p))$$

Introducing residual and Jacobian of the forward problem:

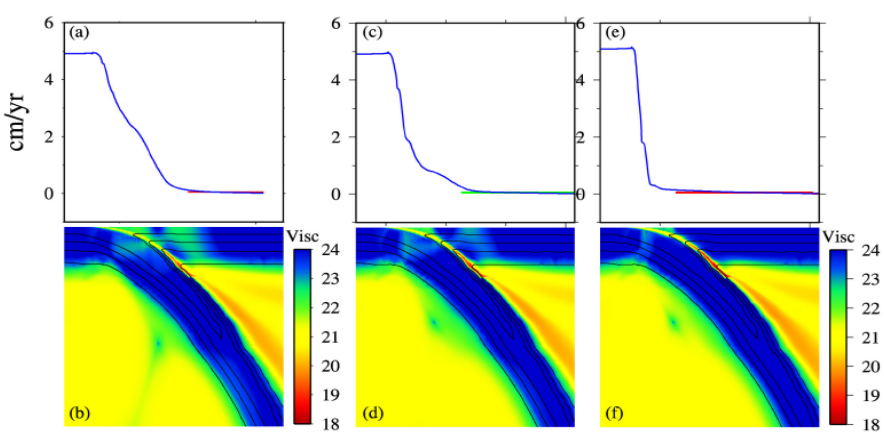
$$R(x) = 0 \qquad J = \frac{\partial R}{\partial x}$$

we can efficiently evaluate the gradient using the adjoint method:

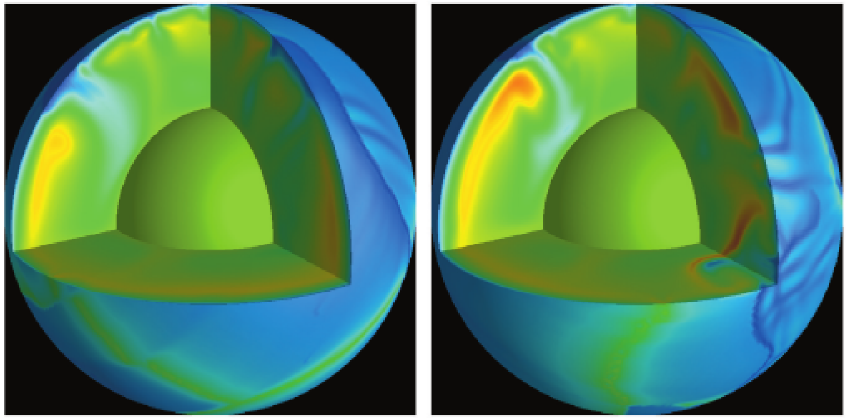
$$G = -\psi^T \frac{\partial R}{\partial p} \qquad \psi = J^{-T} \left( \frac{\partial F}{\partial x} \right)^T$$

which only additionally requires evaluating the derivatives $\frac{\partial R}{\partial p}$ and $\frac{\partial F}{\partial x}$

Widely used technique in geodynamic model community (but not only), e.g.:




Ratnaswamy et al., 2015



Bunge et al., 2003

# Adjoint gradients explained

Objective function, formulated in terms of forward problem solution (x)

$$F(x, x(p))$$

Minimization (optimization, inversion)

Find input parameters (p) such that F assumes minimum (preferably global) value

$$\min_p (F(x, x(p)))$$

Gradient-based methods (steepest descend, BFGS) require calculating the gradient of the objective function w.r.t input parameters

$$G = \frac{dF}{dp}$$

LIMITATION! Gradient methods are not suitable for finding global minima.

# Gradient of objective function

How do we get gradient? Just use chain rule!

$$G = \frac{dF}{dp} = \frac{\partial F}{\partial p} + \frac{\partial F}{\partial x}\frac{dx}{dp}$$

$\frac{\partial F}{\partial p}$ - Easy term (assumed to be zero in what follows)

$\frac{\partial F}{\partial x}$ - Easy term (objective function is usually directly expressed in terms of forward problem solution)

$\frac{dx}{dp}$ - difficult term (so-called (flow) sensitivity parameters)
ONE OF THE MAJOR LIMITATIONS OF ADJOINT METHOD

x should be smooth and differentiable function of p. For certain rheology types (DP plasticity) it is not the case.

# Sensitivity parameters

How to get dx/dp? Use chain rule once again plus an observation!

$$R(x) = 0$$

- residual of the forward problem

$$\frac{dR}{dp} = \frac{\partial R}{\partial p} + \frac{\partial R}{\partial x}\frac{dx}{dp}$$

- derivative of the residual (chain rule)

$$\frac{dR}{dp} = 0$$

- why is that? Simple: forward problem residual must be zero for any set of input parameters. MAJOR TRICK!

$$\frac{\partial R}{\partial x} = J$$

- Jacobian matrix

$$\frac{dx}{dp} = -J^{-1}\frac{\partial R}{\partial p}$$

- solve for sensitivity parameters. VERY EXPENSIVE! (requires one linear solve with Jacobian per input parameter)

# Adjoint system

How to make it less expensive?

$$G = \frac{\partial F}{\partial x}\frac{dx}{dp} = -\frac{\partial F}{\partial x}J^{-1}\frac{\partial R}{\partial p}$$

- plug dx/dp into gradient expression

$$\psi = J^{-T}\left(\frac{\partial F}{\partial x}\right)^T$$

- solve adjoint system. CHEAP! Requires only one solve, but with Jacobian transpose (adjoint)

$$G = -\psi^T\frac{\partial R}{\partial p}$$

- evaluate gradient (check by plugging psi)

$$(AB)^T = B^T A^T$$
$$(A)^{-T} = (A^T)^{-1} = (A^{-1})^T$$

The major advantage of adjoint method is that it requires only one linear solve per gradient evaluation. The disadvantage is that this solve involves Jacobian transpose. Symmetric cases are insensitive, but certain rheology types (DP plasticity) and discretizations (FDSTAG) are sensitive.

# Residual derivatives

The only remaining term is the derivative of forward problem residual:

$$\frac{\partial R}{\partial p}$$

For each new type of the input parameter (e.g. density, power-law exponent) it can be obtained by directly differentiating the residual expressions.

Alternatively one can use finite differences (for each input parameter):

$$\frac{\partial R}{\partial p_i} = \frac{R(p + e_i h) - R(p)}{h}$$

The derivatives are normally very sparse vectors, since only limited number of residual components are affected by each input parameter.

Irrespective of the evaluation method, one should utilize the sparsity.

# Adjoint scaling law

Scaling law relates change in the observable with the change in the solution parameter.

Consider general multi(two)-parametric scaling law:

$$q = a\, x^b y^c$$

Exponents can be determined separately by taking derivatives:

$$\frac{\partial q}{\partial x} = a\; b\; x^{b-1} y^c = \frac{b\, q}{x}$$

Which can be rearranged as (provided that gradients are known):

$$b = \frac{\partial q}{\partial x} \frac{x}{q}$$

We can view the observable (q) as an objective function and compute adjoint gradients!

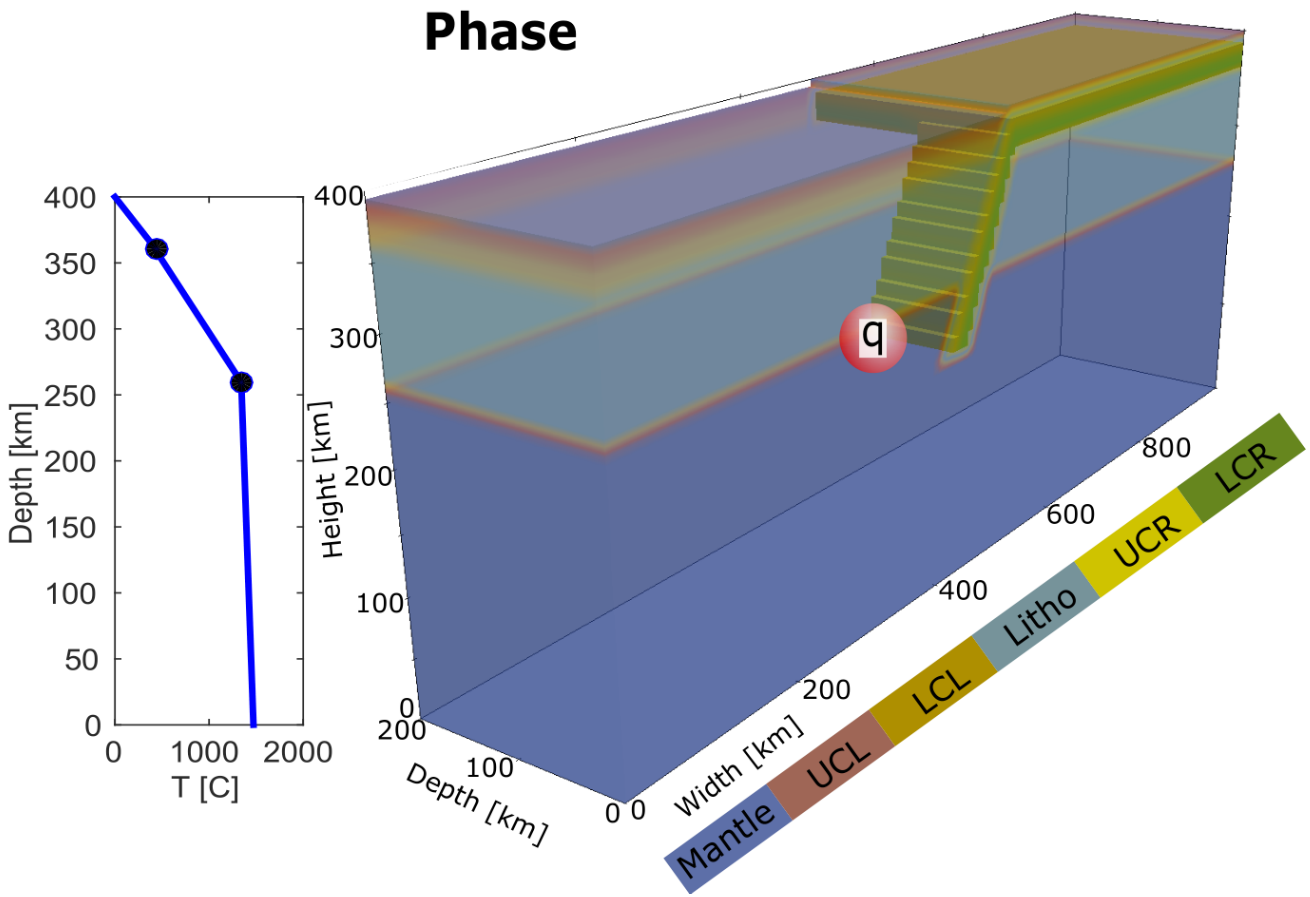
The remaining step is finding the prefactor:

$$a = \frac{q}{x^b y^c}$$

# 3D subduction synthetic test

Adjoint gradient evaluation is implemented in LaMEM (Reuber et al., in preparation)
LaMEM is integrated with TAO package (Toolkit for Advanced Optimization)

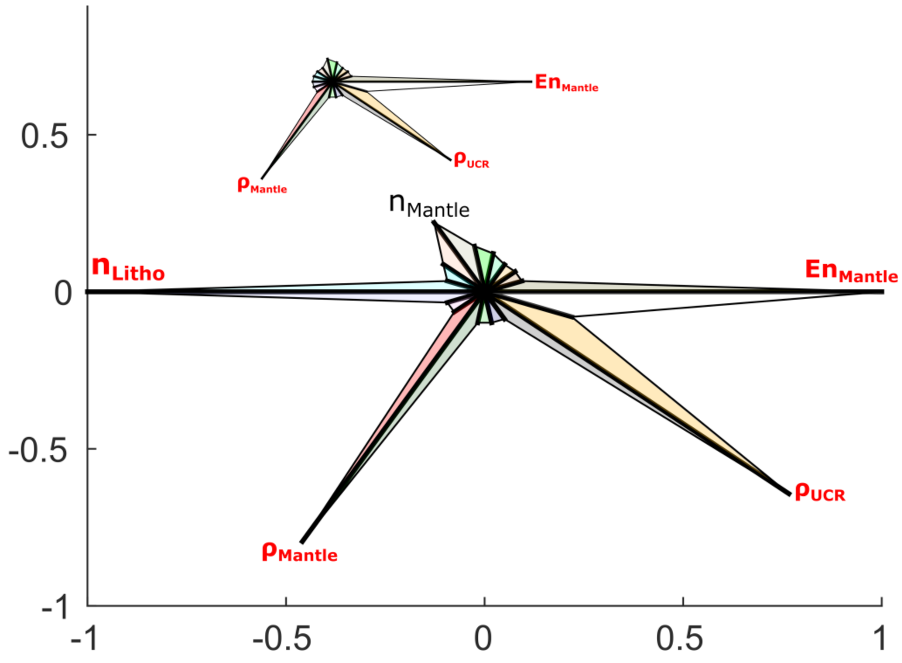
More info is on poster of Georg Reuber




Growth rate at point q is an observable.
Activation energy and power law exponent are the scaling law parameters.

# Adjoint scaling law result

Normalized polar plot of scaling law exponents



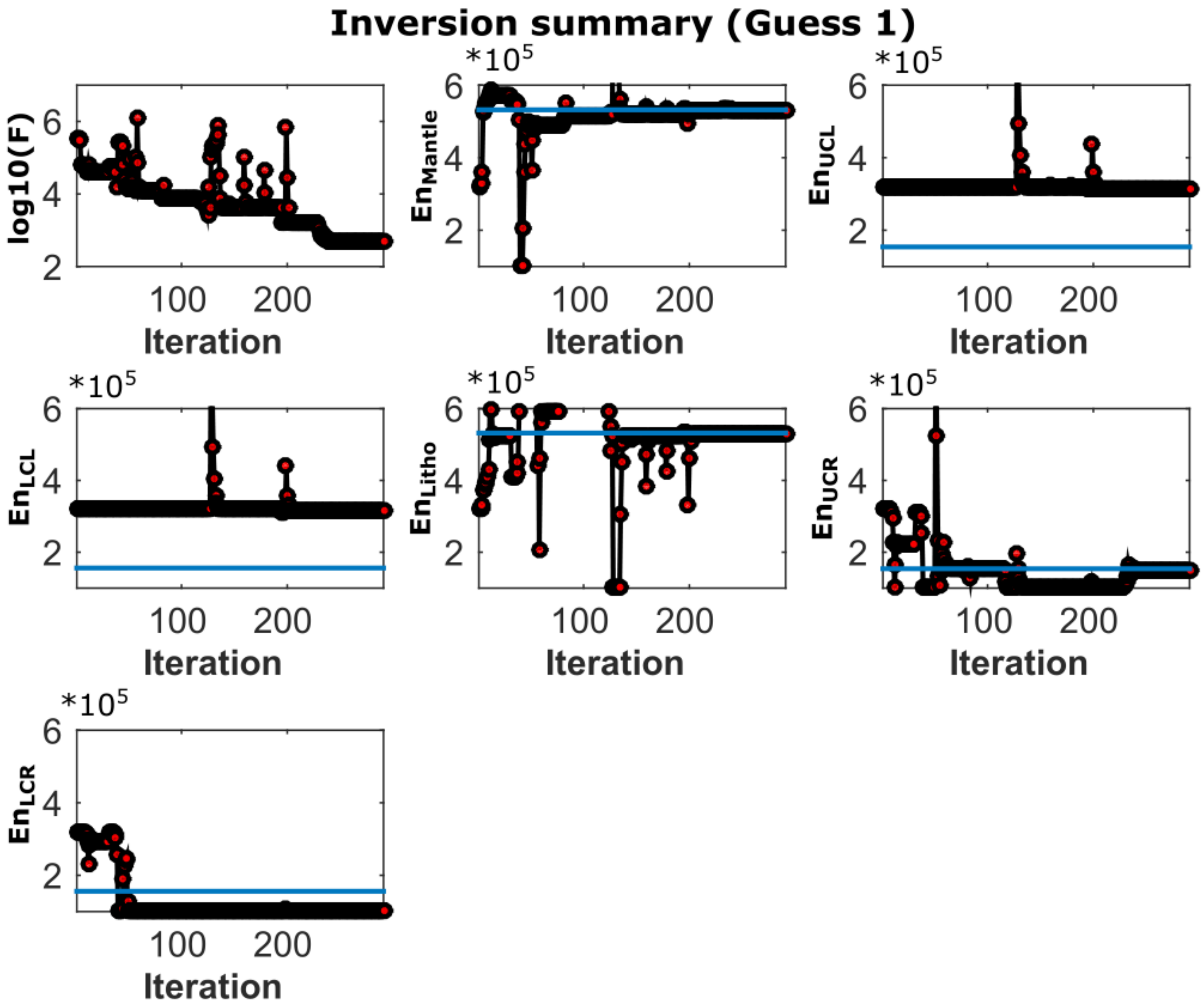

**Scaling law**

$$9.5 = 6.3e123 * 532000^{-18.2} * 154000^{0} * 156000^{0} * 532000^{-1} * 154000^{-2.2} * 256000^{-2.7} * 2.3^{0} * 2.4^{0} * 3.5^{0} * 2.3^{0} * 2.4^{0} * 3300^{13} * 2700^{-5e-4} * 3400^{-0.004} * 3300^{1.1} * 2700^{-14} * 3400^{3.3}$$

---

$$q = x * En_{Mantle}^{a} * En_{UCL}^{b} * En_{LCL}^{c} * En_{Litho}^{d} * En_{UCR}^{e} * En_{LCR}^{f} * n_{UCL}^{g} * n_{LCL}^{h} * n_{Litho}^{i} * n_{UCR}^{j} * n_{LCR}^{k} * \rho_{Mantle}^{l} * \rho_{UCL}^{m} * \rho_{LCL}^{n} * \rho_{Litho}^{o} * \rho_{UCR}^{p} * \rho_{LCR}^{q}$$

# Adjoint inversion result



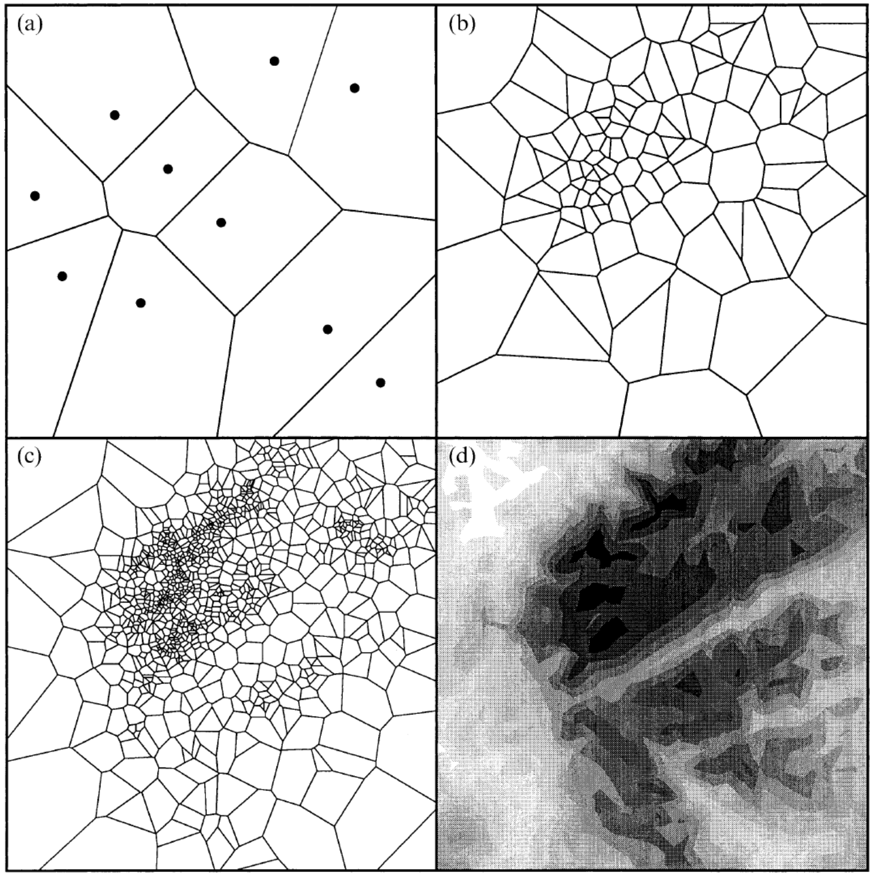

Inversion results for 3D subduction test.
Not all the solution parameters are correctly inverted.
Gradient-based methods are non-unique!

# Neighborhood algorithm (NA) (Sambridge, 1999)

Similar to simulated annealing, genetic and Monte-Carlo algorithms

Builds piecewise-constant Voronoi interpolant of the misfit function

Refines by performing a uniform random walk within lowest misfit cells




Sambridge, 1999

ADVANTAGES:

- *Derivative-free*
- *Samples misfit function*
- *Attempts to identify multiple minima*

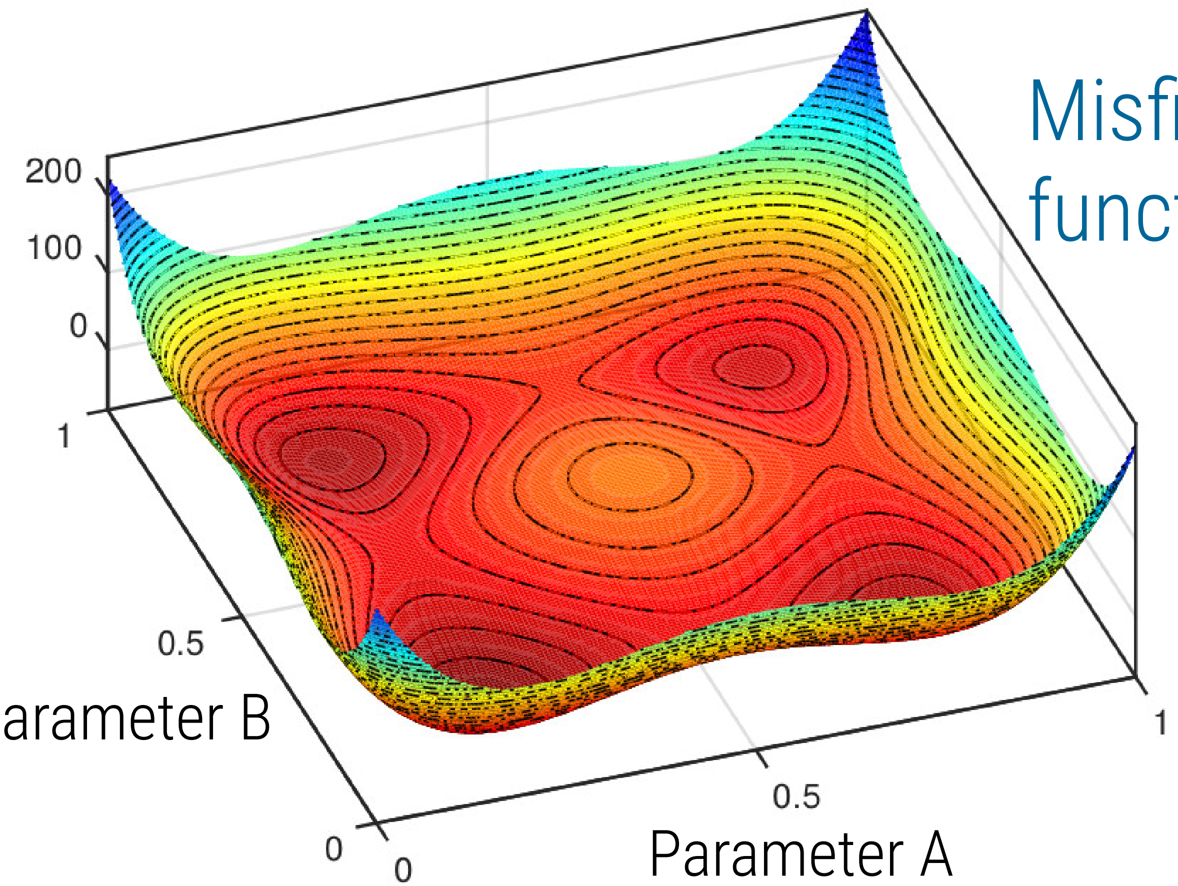
DISADVANTAGES:

- *Requires many forward models*
- *Limited number of parameters*

# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)

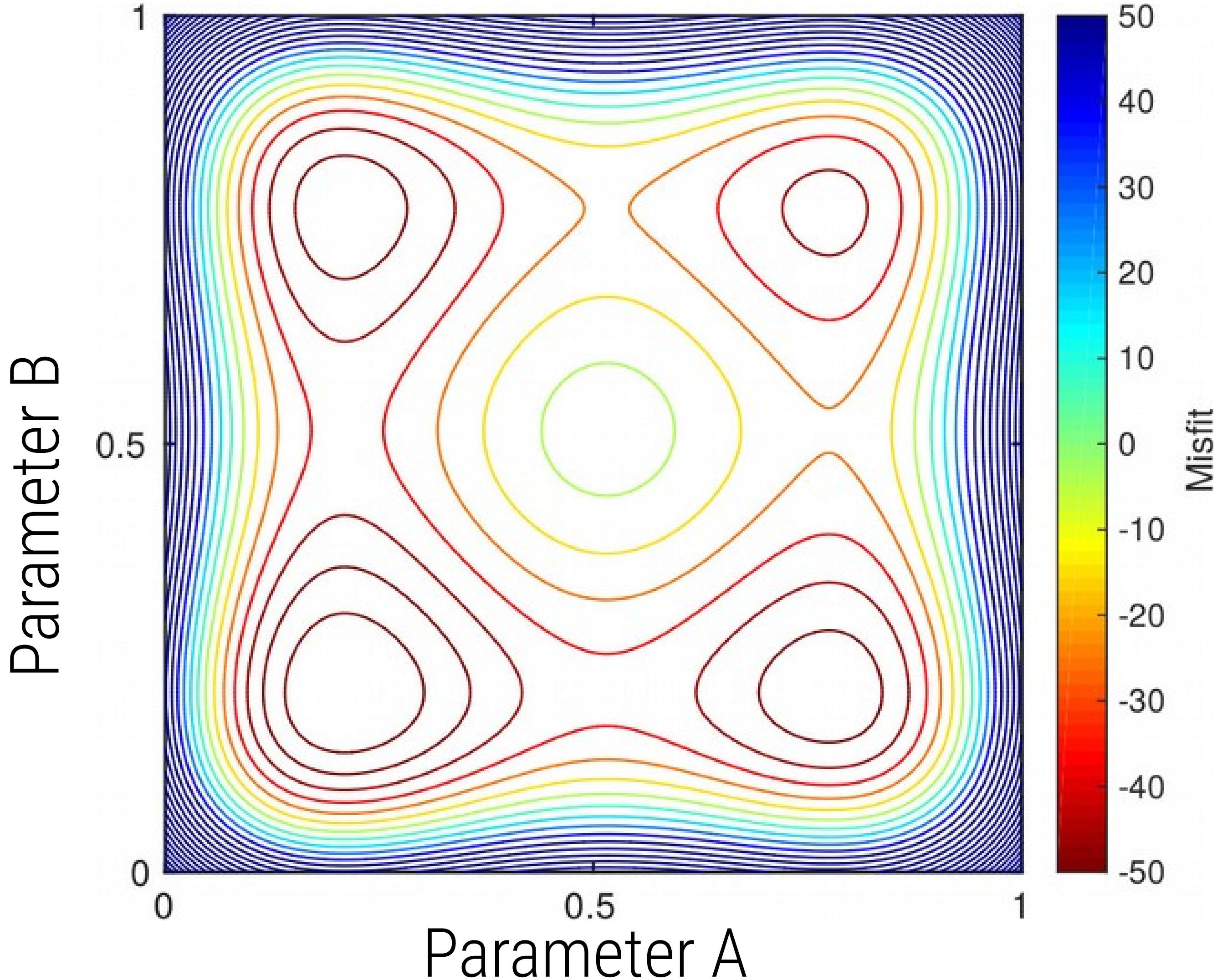



Misfit (objective) function

# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

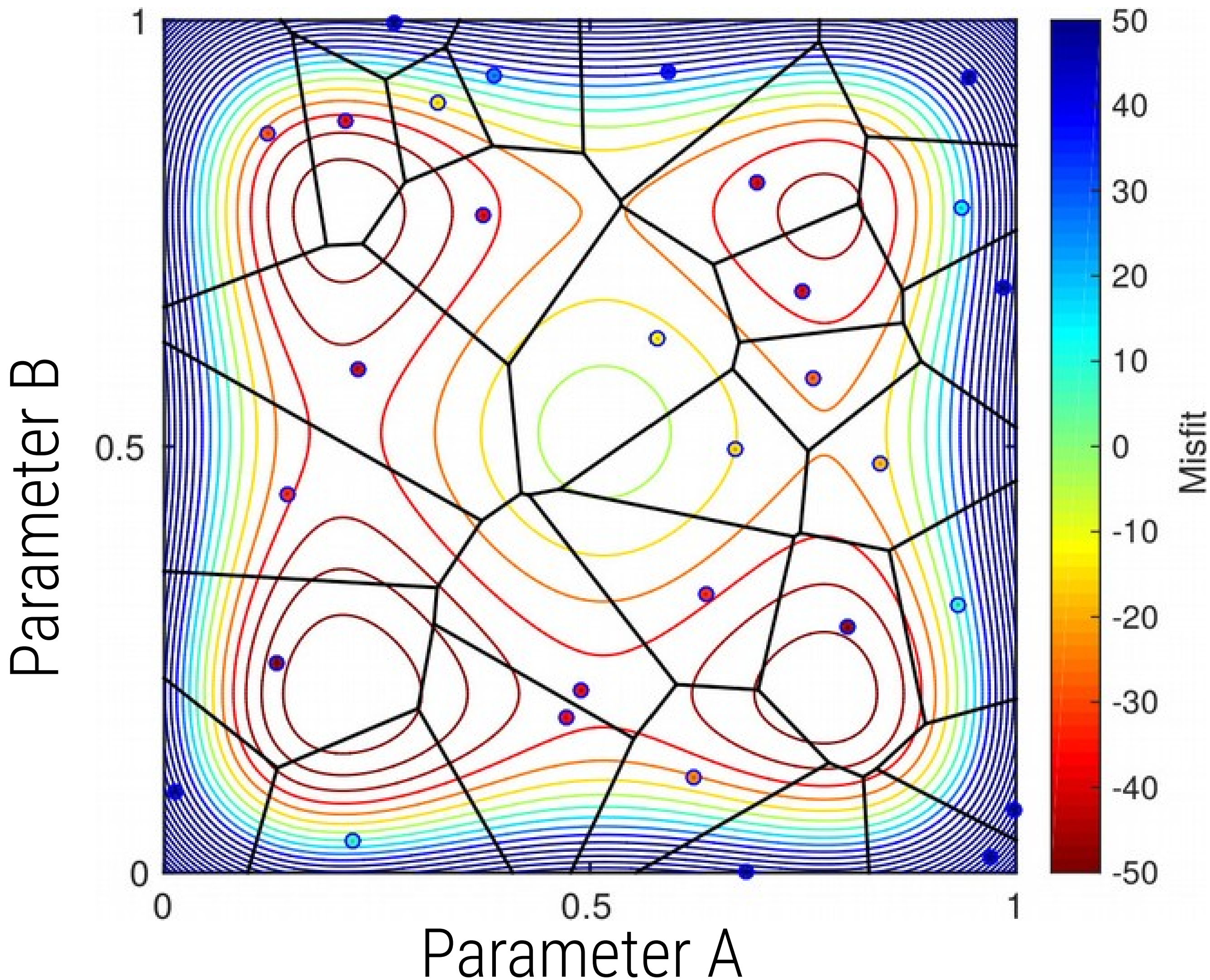
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

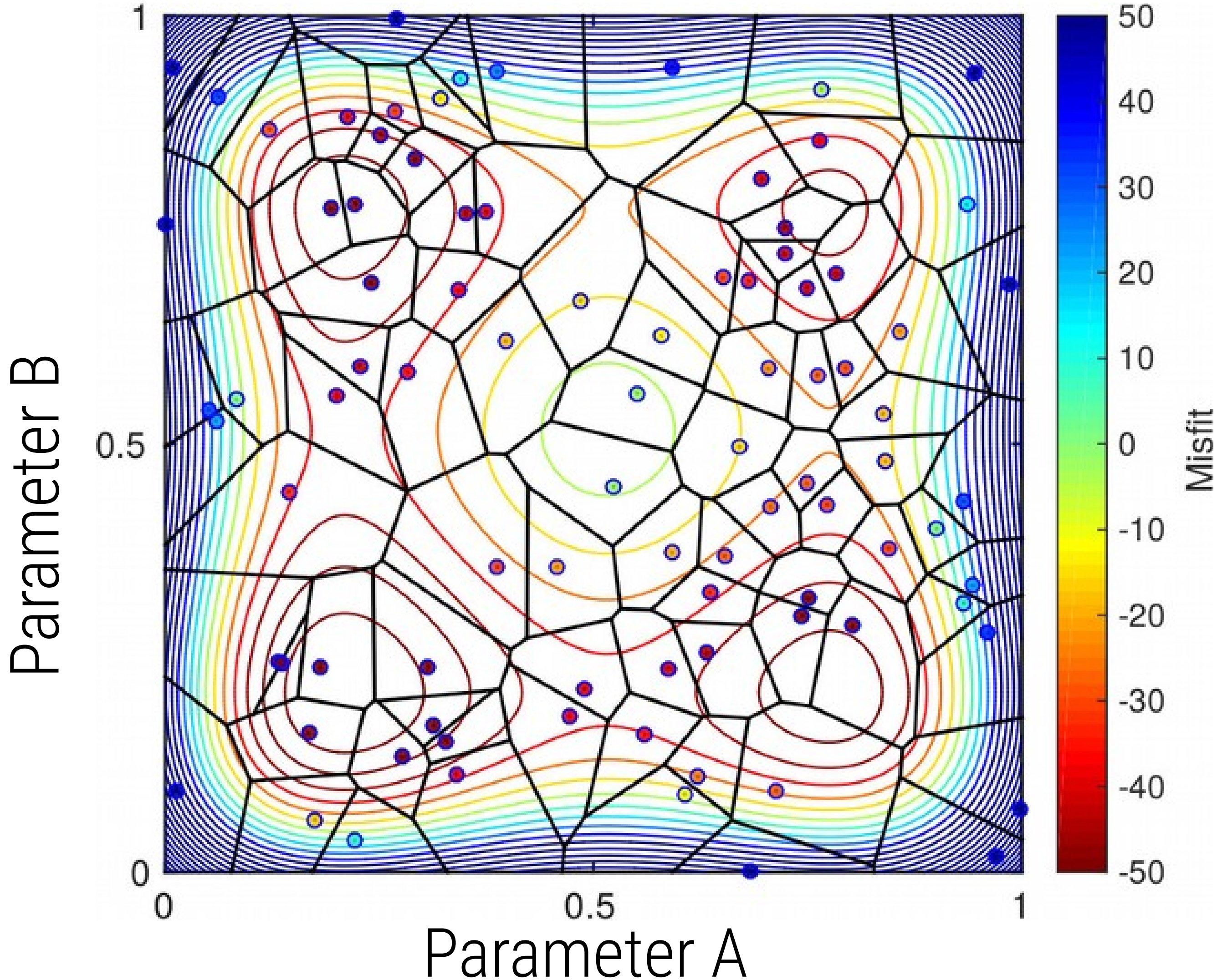
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

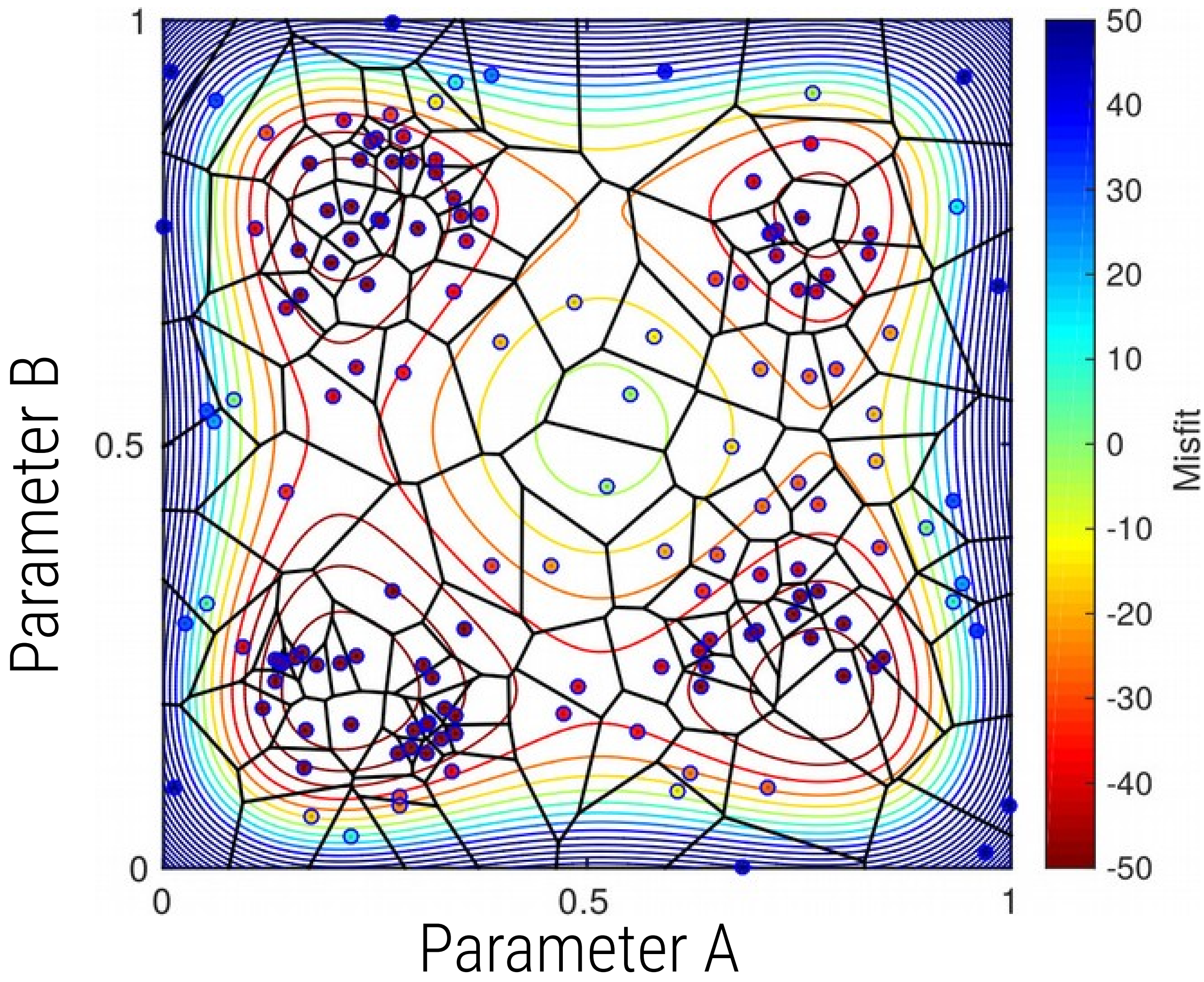
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

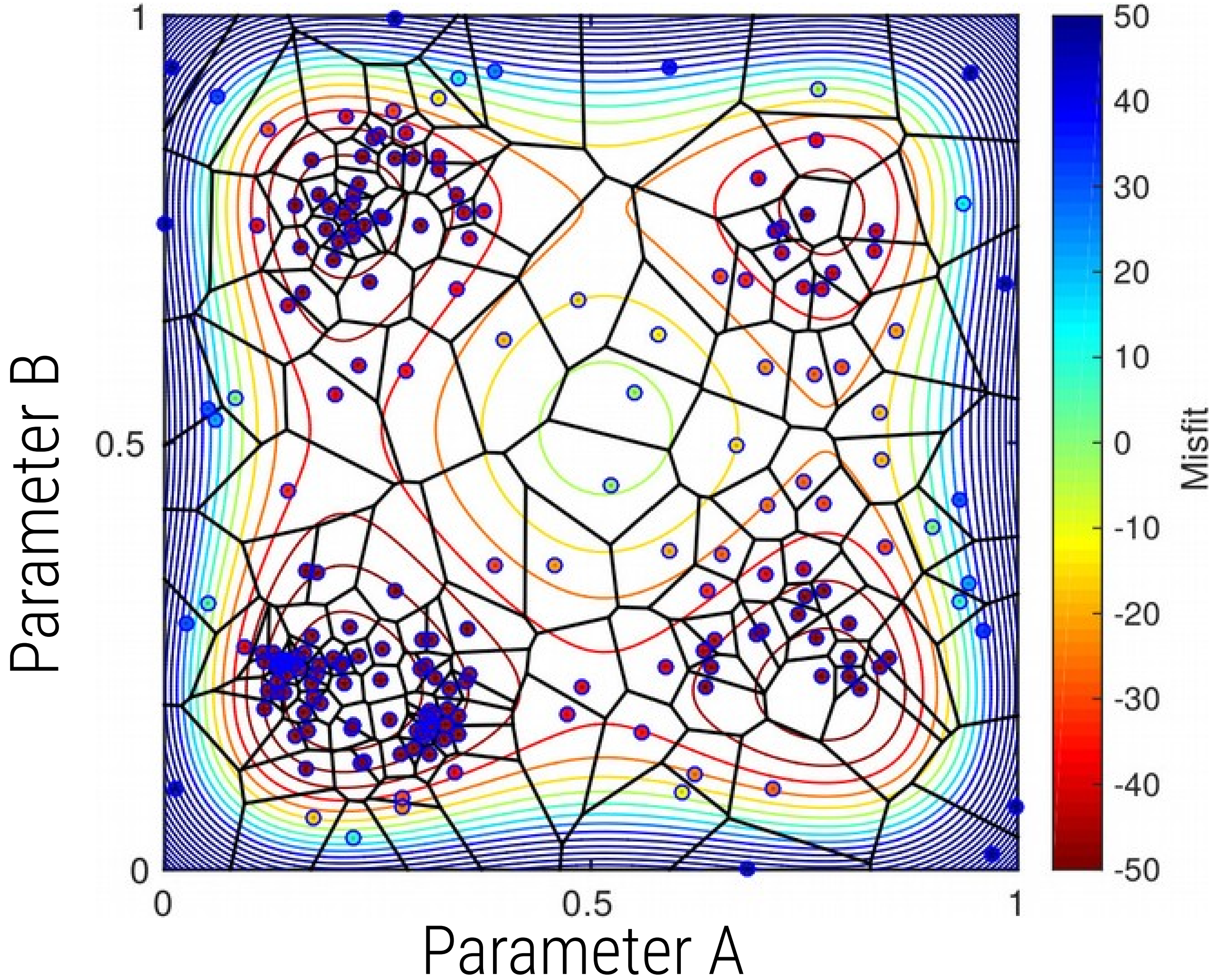
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Neighborhood algorithm

Smarter sampling than Monte Carlo: The Neighborhood Algorithm

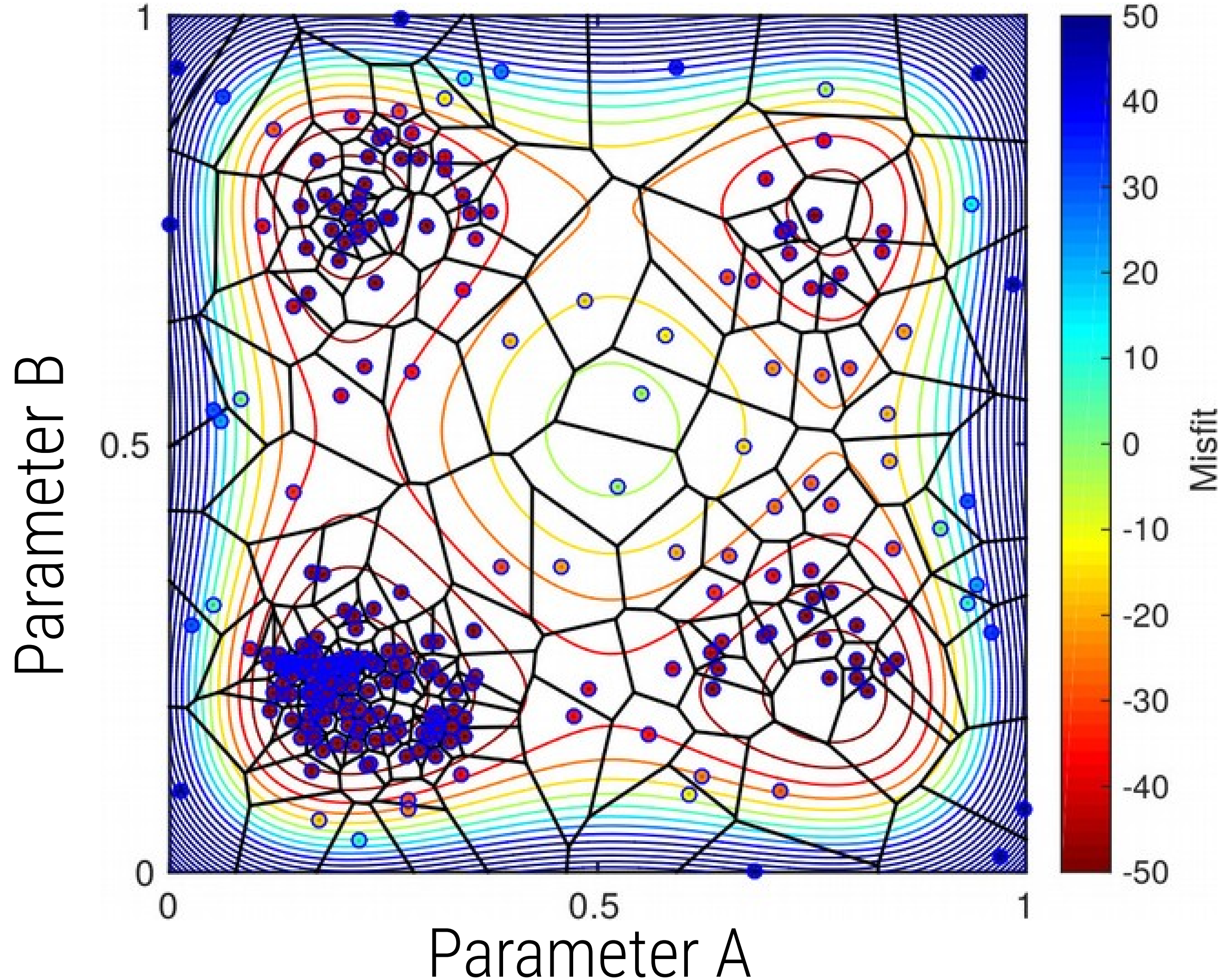
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Neighborhood algorithm

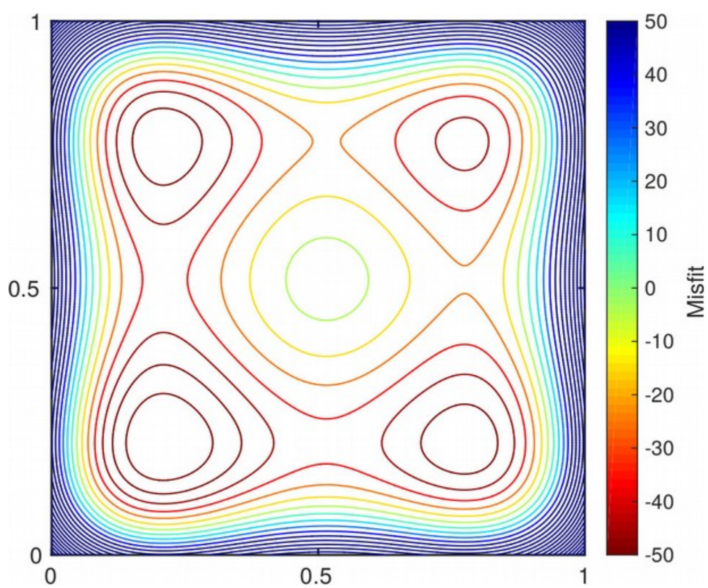
Smarter sampling than Monte Carlo: The Neighborhood Algorithm

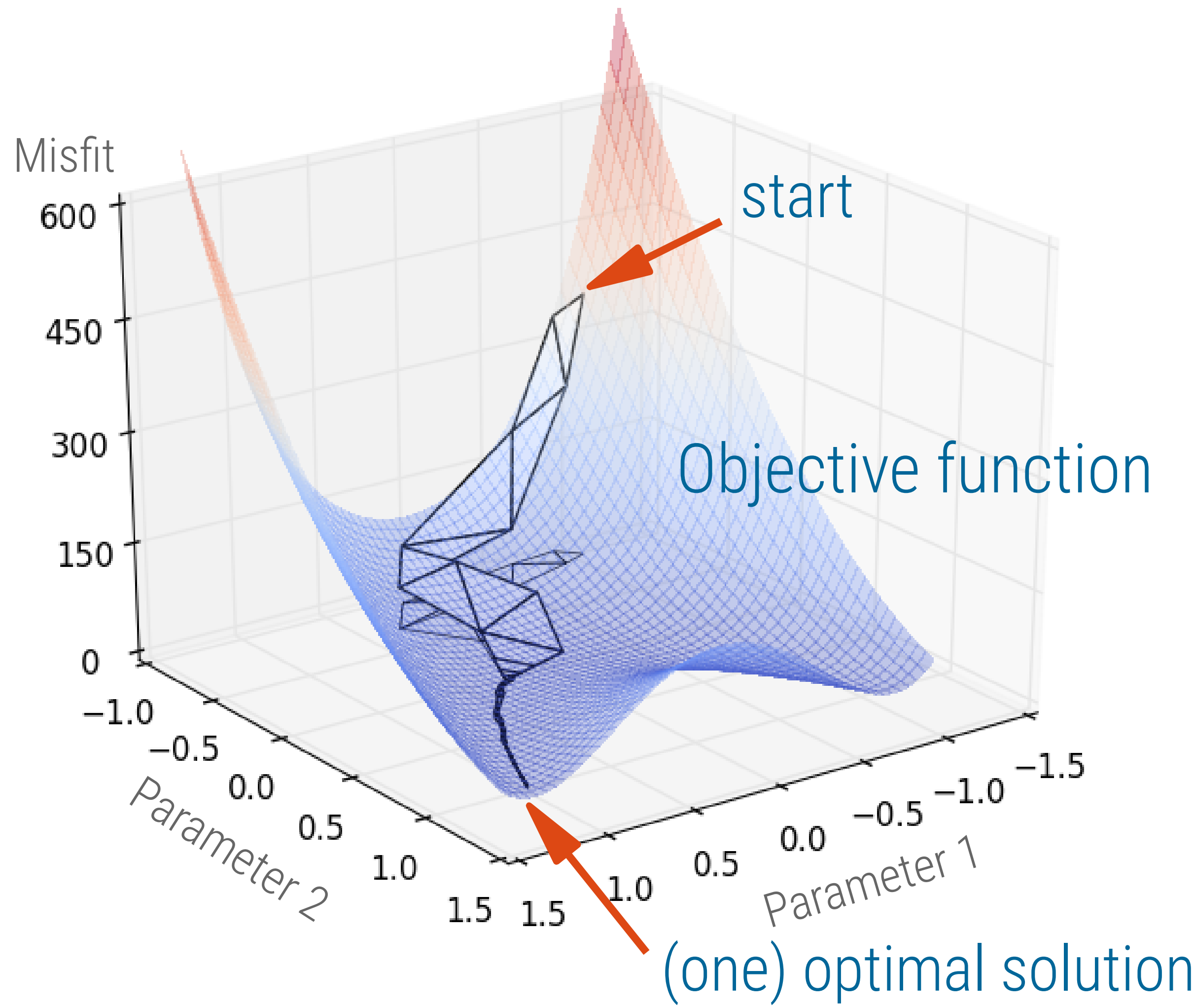
(original: Sambridge, 1999, extended parallel version: Baumann et al. 2014)



# Downhill Simplex Method (Nelder & Mead 1965)

- Integrated with LaMEM using TAO package
- Always replace worst vertex of simplex
  - High dimensions

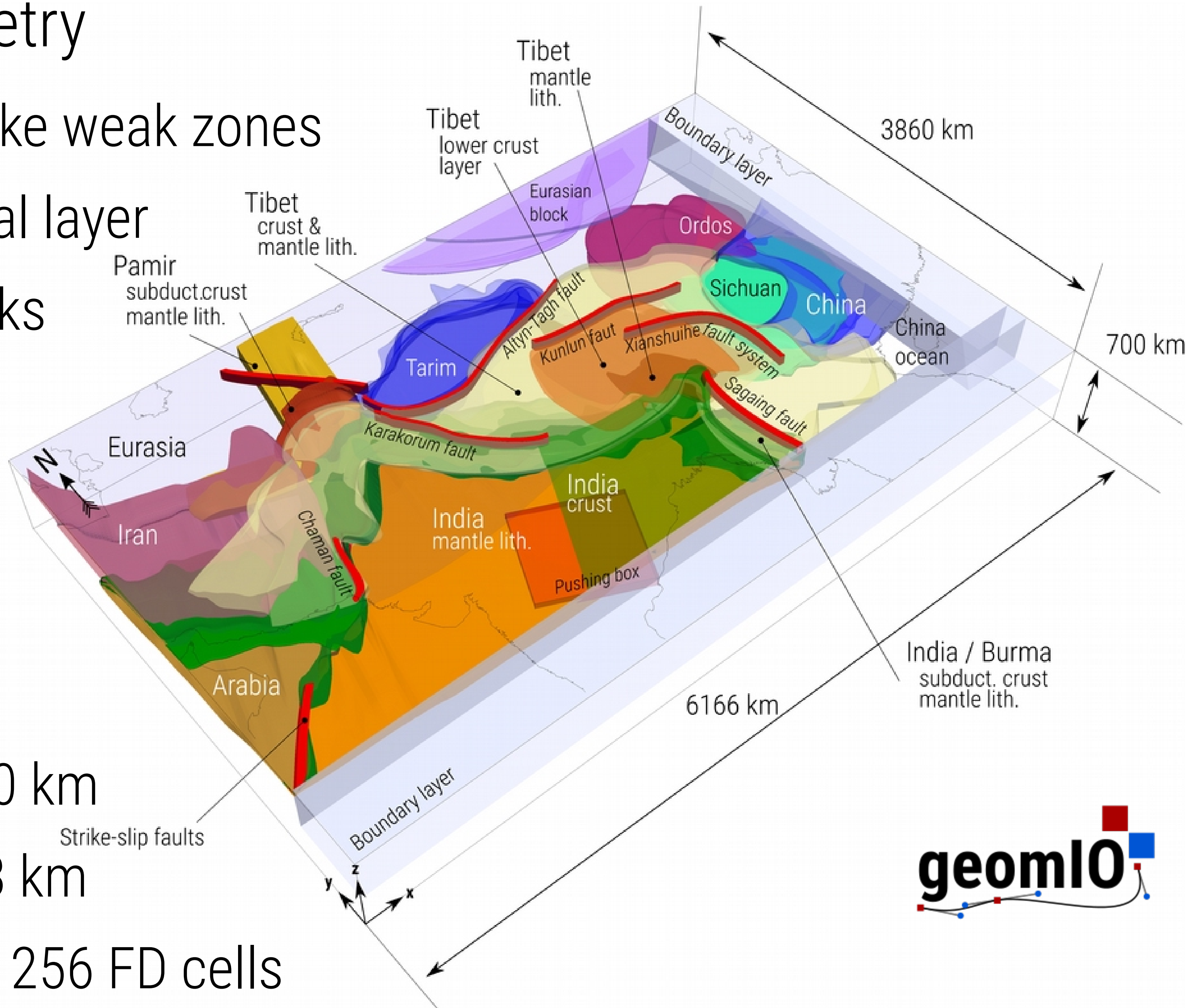


# 3D India-Asia collision

- Model geometry
  - Strike-slip like weak zones
  - Weak crustal layer
  - Strong blocks
  - Slabs

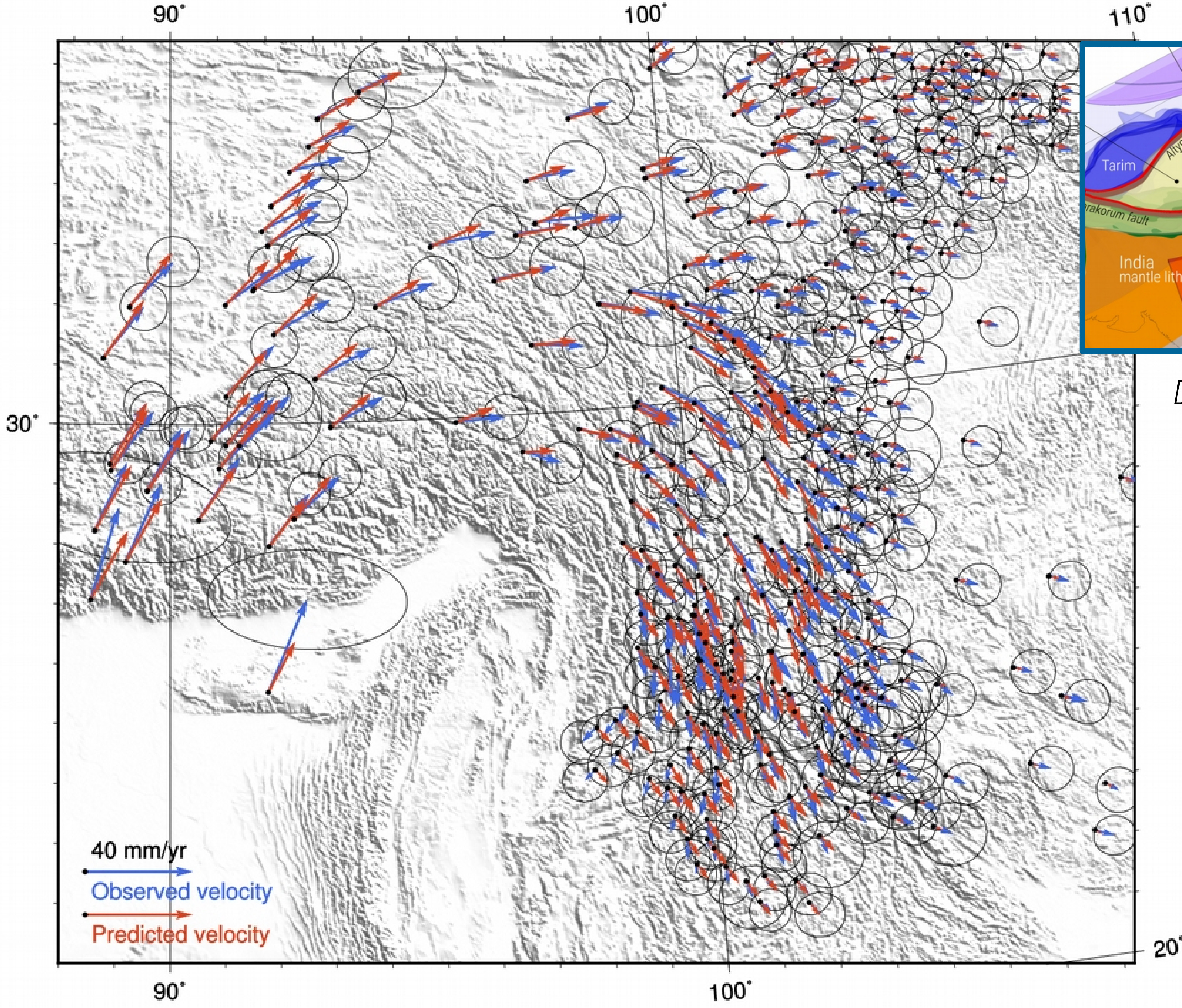
- Resolution
  - Lateral: 7-10 km
  - Vertical: 2-3 km
  - 512 x 512 x 256 FD cells

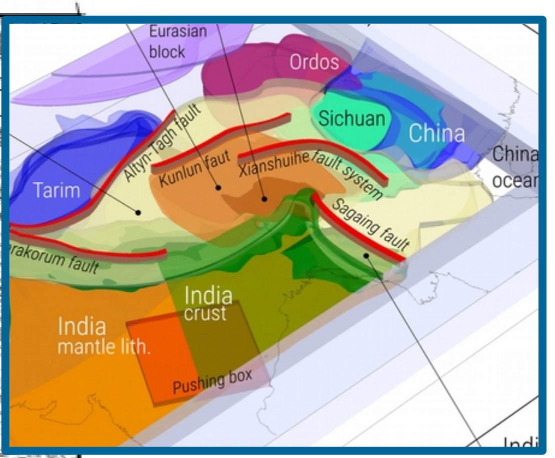



geomIO

Baumann et al. (in preparation)

# 3D India-Asia collision




*Data: Gan et al. (2007)*

"Best fit" model

# 3D India-Asia collision

"Best fit" model