Tutorial Handbook Part 2

Bonus exercises

Congratulations. You finalized your Basic Scheme. The following tasks may be accomplished in indefinite order and you can chose the task you like. Furthermore, you may get more creative with your programming style.

Contents

1 Acceleration

This exercise is concerned with speeding up your code so you can handle larger geometries. Complete the following tasks:

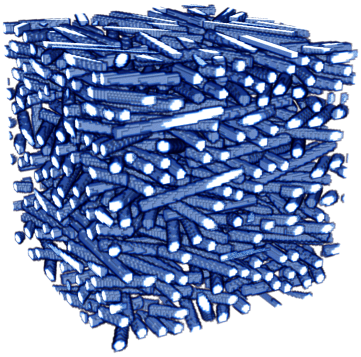
1. Consider the `ball_32.mha` microstructure and monitor your code using a timing library such as `time`. Which task requires the most computation time?

2. Typically, large loops (for instance looping over all voxels) require a lot of computation time. These loops are rather slow in Python, as it is an interpreted language. In order to speed up these loops, *Cython* enables you to write C extensions for Python using a syntax that is very similar to Python. These are then compiled beforehand. Some hints are in order:

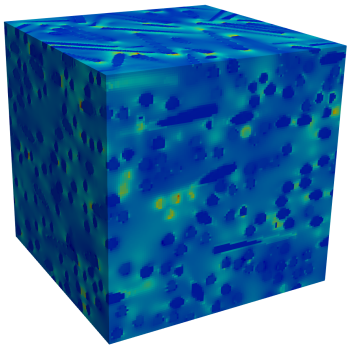
   - Start by writing code the same way you would in Python. You can then "cythonize" the parts of your code you deem too slow.
   - You can improve your performance significantly by adding static type declarations (see the small example in the `CythonExample` folder).
   - If you want to run your Cython code, you will have to compile it first. You tell Cython what to compile by writing a small `setup.py` file. Then, run the command `python setup.py build_ext --inplace` in your terminal to compile your code. After compilation, you can import the Cython function the same way you would import normal Python libraries.
   - During the compilation, an `.html` file is generated. Check that file for further optimization of your code. The file highlights all parts with Python interactions. In order to speed up your code, you should try to avoid looping over parts with a lot of Python interactions.

   Write Cython extensions to replace your "slow" code, compile them and monitor the speedup of your code.

3. With your *fast* code at hand you can investigate larger geometries. Investigate the fiber reinforced composite structures (in the `Microstructures/Fibers` folder) and their solution fields.



(a) Fiber reinforced composite discretized with 128^3 voxels.



(b) ε_{xx} component of the strain under macroscopical uniaxial strain state.

2 Additional discretizations

2.1 Introduction

This additional exercise is concerned with implementing additional Γ^0-operators. The Γ^0-operators allow you to encode different discretization types, such as Finite Difference or Finite Element discretizations. Whereas the previously implemented operator is based on trigonometric collocation, additional finite-difference based operators are available.

Task Implement the two additional discretization methods. Update your code so that you can easily switch between the three different discretization methods. If you already accomplished the previous task on acceleration be sure to stick to Cython for your additional operators.
Test your novel discretizations with an infinite material contrast. Use the `ball_32.mha` microstructure and set the Young's modulus of the inclusion to 0. What do you notice in the iteration count of your solver for the different discretization choices?

2.2 The Moulinec-Suquet discretization

Let us start with a recap of the implementation of the Moulinec-Suquet discretization you already realized. First, we precompute the frequency vectors

```
xfreq = np.fft.fftfreq(N_x)
yfreq = np.fft.fftfreq(N_y)
zfreq = np.fft.rfftfreq(N_z)
```

With these at hand, the application of the (negative) operator $-\Gamma^0$ in position $[i, j, k]$ may be efficiently computed via Algorithm 1. In this form, we normalize the frequency vector to norm 1 and compute matrix-vector multiplications and outer products efficiently. Pay attention to Voigt's notation.

Algorithm 1 In-place application of $-\Gamma^0$ for the Moulinec-Suquet discretization

1: $\xi \leftarrow \begin{pmatrix} \text{xfreq[i]} \\ \text{yfreq[j]} \\ \text{zfreq[k]} \end{pmatrix}$
2: $\eta \leftarrow \xi / \|\xi\|$
3: $\tau \leftarrow \texttt{field}[:, i, j, k]$
4: $f \leftarrow \tau \cdot \eta$
5: $s \leftarrow f \cdot \eta$
6: $u \leftarrow (-f + \frac{1}{2} s\eta)/\mu_0$
7: $\varepsilon \leftarrow 0.5(\eta \otimes u + u \otimes \eta)$
8: $\texttt{field}[:, i, j, k] \leftarrow \varepsilon$

2.3 The rotated staggered grid discretization

One popular (reduced integration) Finite Element discretization operates on a rotated staggered grid. In order to apply the non-local Γ-operator, it is convenient to precompute both the frequency vectors

```
xfreq = np.fft.fftfreq(N_x)
yfreq = np.fft.fftfreq(N_y)
zfreq = np.fft.rfftfreq(N_z)
```

and, additionally, the following complex expressions

```
e_x = np.exp(2j * np.pi * xfreq)
e_y = np.exp(2j * np.pi * yfreq)
e_z = np.exp(2j * np.pi * zfreq)
```

The evaluation of the operator is performed in the following algorithm, where $\bar{z}$ denotes the complex conjugate of the complex number z. Once again, pay attention to Voigt's notation.

Algorithm 2 In-place application of $-\Gamma^0$ for the rotated staggered grid discretization

1: $\xi \leftarrow \begin{pmatrix} (\text{e\_x[i]} + 1)\ (\text{e\_y[j]} - 1)\ (\text{e\_z[k]} - 1) \\ (\text{e\_x[i]} - 1)\ (\text{e\_y[j]} + 1)\ (\text{e\_z[k]} - 1) \\ (\text{e\_x[i]} - 1)\ (\text{e\_y[j]} - 1)\ (\text{e\_z[k]} + 1) \end{pmatrix}$
2: $\eta \leftarrow \xi / \|\xi\|$
3: $\tau \leftarrow \texttt{field}[:, i, j, k]$
4: $f \leftarrow \tau \cdot \bar{\eta}$
5: $s \leftarrow f \cdot \bar{\eta}$
6: $u \leftarrow (-f + \frac{1}{2} s\eta)/\mu_0$
7: $\varepsilon \leftarrow 0.5(\eta \otimes u + u \otimes \eta)$
8: $\texttt{field}[:, i, j, k] \leftarrow \varepsilon$

2.4 The staggered grid discretization

A different approach operates on the standard (non-rotated) staggered grid. This discretization treats shear componets and normal components differently, and some caution is advised. Similar precomputations to the rotated staggered grid discretization are required, i.e., the frequencies

```
xfreq = np.fft.fftfreq(N_x)
yfreq = np.fft.fftfreq(N_y)
zfreq = np.fft.rfftfreq(N_z)
```

should be used to compute

```
k_x = np.exp(-2j * np.pi * xfreq)
k_y = np.exp(-2j * np.pi * yfreq)
k_z = np.exp(-2j * np.pi * zfreq)
```

Algorithm 3 In-place application of $-\Gamma^0$ for the staggered grid discretization

1: $\xi \leftarrow \begin{pmatrix} \mathrm{k\_x[i]} - 1 \\ \mathrm{k\_y[j]} - 1 \\ \mathrm{k\_z[k]} - 1 \end{pmatrix}$

2: $\eta \leftarrow \xi / \|\xi\|$

3: $\tau \leftarrow \mathtt{field}[:, i, j, k]$

4: $f \leftarrow \begin{pmatrix} -\hat{\tau}_{11}\eta_1 + \hat{\tau}_{12}\bar{\eta}_2 + \hat{\tau}_{13}\bar{\eta}_3 \\ \hat{\tau}_{21}\bar{\eta}_1 - \hat{\tau}_{22}\eta_2 + \hat{\tau}_{23}\bar{\eta}_3 \\ \hat{\tau}_{31}\bar{\eta}_1 + \hat{\tau}_{32}\bar{\eta}_2 - \hat{\tau}_{33}\eta_3 \end{pmatrix}$

5: $s \leftarrow f \cdot \bar{\eta}$

6: $u \leftarrow (-f + \frac{1}{2} s\eta)/\mu_0$

7: $\varepsilon \leftarrow \begin{pmatrix} -u_1\bar{\eta}_1 & \frac{u_1\eta_2 + u_2\eta_1}{2} & \frac{u_1\eta_3 + u_3\eta_1}{2} \\ \frac{u_1\eta_2 + u_2\eta_1}{2}\bar{\eta}_1 & -u_2\bar{\eta}_2 & \frac{u_3\eta_2 + u_2\eta_3}{2} \\ \frac{u_1\eta_3 + u_3\eta_1}{2} & \frac{u_3\eta_2 + u_2\eta_3}{2} & -u_3\bar{\eta}_3 \end{pmatrix}$

8: $\mathtt{field}[:, i, j, k] \leftarrow \varepsilon$

3 Fast solvers

3.1 The Barzilai-Borwein method

The implemented Basic Scheme may be interpreted as a gradient descent method with constant step size $1/\alpha^0$. The iteration reads

$$\varepsilon^{k+1} = \bar{\varepsilon} - \frac{1}{\alpha^0}\,\Gamma : (\mathbb{S}(\cdot, \varepsilon^k) - \alpha^0\ \varepsilon^k).$$

A significant speedup may be achieved using an adaptive step size strategy

$$\varepsilon^{k+1} = \bar{\varepsilon} - \frac{1}{\alpha^k}\,\Gamma : (\mathbb{S}(\cdot, \varepsilon^k) - \alpha^k\ \varepsilon^k),$$

where α^k is chosen in every iteration. One promising approach, the Barzilai-Borwein method, computes the step size via

$$\alpha^k = \frac{\langle \sigma(\varepsilon^k) - \sigma(\varepsilon^{k-1}), \varepsilon^k - \varepsilon^{k-1} \rangle_{L^2}}{\| \varepsilon^k - \varepsilon^{k-1} \|_{L^2}^2},$$

which is evaluated before computing the polarization. For the first iteration, the standard reference material can be used.

Task Implement the Barzilai-Borwein method (Hint: Voigt notation!) by updating your reference material to accept a varying α^k. Compare the iteration count with that of the Basic Scheme.

3.2 The Conjugate Gradient method

An optimal solver for linear problems is given by the CG method:

Algorithm 4 Linear CG $(\mathbb{C}, \bar{\varepsilon}, \text{maxit}, \texttt{tol})$

$G \leftarrow \Gamma : \mathbb{C} : \bar{\varepsilon}$ ▷ Compute $\bar{\sigma} = \langle \mathbb{C} : \bar{\varepsilon} \rangle_Q$
$D \leftarrow -G$
$[\texttt{r}, \texttt{res}] \leftarrow [\|G\|, \|G\|/\|\bar{\sigma}\|]$
$k \leftarrow 0$
while $k <$ maxit **and** $\texttt{res} > \texttt{tol}$ **do**
  $k \leftarrow k + 1$
  $\texttt{r}_{\text{old}} \leftarrow \texttt{r}$
  $Z \leftarrow \Gamma : \mathbb{C} : D$ ▷ $[\triangle\bar{\sigma}, \hat{Z}(0)] \leftarrow [\hat{Z}(0), 0]$
  $\alpha \leftarrow \texttt{r}^2/(D, Z)_{L^2}$
  $\varepsilon \leftarrow \varepsilon + \alpha\, D$
  $\bar{\sigma} \leftarrow \bar{\sigma} + \alpha\, \triangle\bar{\sigma}$
  $\texttt{res} \leftarrow r/\|\bar{\sigma}\|$
  $G \leftarrow G + \alpha\, Z$
  $\texttt{r} \leftarrow \|G\|$
  $\gamma \leftarrow \texttt{r}^2/\texttt{r}^2_{\text{old}}$
  $D \leftarrow -G + \gamma\, D$
end while
$\varepsilon \leftarrow \varepsilon + \bar{\varepsilon}$
return $\varepsilon, \bar{\sigma}$

Task Implement the CG method. In addition to your strain field, you need to keep the fields D, Z and G in memory. Furthermore, set your reference material $\alpha^0 = 1$ so you do not have to change your $\Gamma-$operator. Compare the CG method with the Basic Scheme.

4 Additional physics – thermal conductivity

Within the previously implemented framework of solving the Lipmann-Schwinger equation for elasticity, additional physical settings may be studied. For thermal conductivity problems, we consider the temperature gradient $s \in \mathbb{R}^3$ with $\langle s \rangle_Q = \bar{s}$, as well as the heat flux $q = As$ with heat conductivity A. For simplicity, let us restrict to an isotropic conductivity, i.e., $A(x) = \kappa(x)$ Id. The Basic Scheme for heat conductivity reads

$$s^{k+1} = \bar{s} - \Gamma^0(As^k - A^0 s^k)$$

and looks very similar to the one for elasticity. In order to change the 'physics' of your code you need the following incredients:

- Your Field should be of dimension $3 \times N_x \times N_y \times N_z$
- An isotropic material law is applied as $q = \kappa s$.
- The reference material of a two-phase material computes as $\alpha^0 = 0.5(\kappa^0 + \kappa^1)$.
- There is no need for Voigt/Mandel notation, which simplifies matters ...
- Your non-local operator Γ^0 reduces to

Algorithm 5 In-place application of $-\Gamma^0$ for conductivity problems

1: $\xi \leftarrow \begin{pmatrix} \text{xfreq[i]} \\ \text{yfreq[j]} \\ \text{zfreq[k]} \end{pmatrix}$
2: $\eta \leftarrow \xi / \|\xi\|$
3: $\tau \leftarrow \texttt{field}[:, i, j, k]$
4: $f \leftarrow \tau \cdot \eta / \alpha^0$
5: $\texttt{field}[:, i, j, k] \leftarrow -f\eta$

- All other operations, such as the Material Applicator or your solvers, should work without any adjustments.