✅Recreating ISMIP HOM - Pattyn 2008

Pattyn, F., Perichon, L., Aschwanden, A., Breuer, B., de Smedt, B., Gagliardini, O., Gudmundsson, G. H., Hindmarsh, R. C. A., Hubbard, A., Johnson, J. V., Kleiner, T., Konovalov, Y., Martin, C., Payne, A. J., Pollard, D., Price, S., Rückamp, M., Saito, F., Souček, O., Sugiyama, S., and Zwinger, T.: Benchmark experiments for higher-order and full-Stokes ice sheet models (ISMIP-HOM), The Cryosphere, 2, 95-108, doi:10.5194/tc-2-95-2008, 2008

Similarities for Experiments A,B,C and D

Table 1- Physical parameters

yts = 31556926 # s a^{-1}
A = 1e-16 # Pa^-n a^{-1}
-> A_sec = A / yts
rho = 910 # kg m^-3
g = 9.81 m s^-2
n = 3

L = [160, 80, 40, 20, 10, 5] # km
H = 1000 # m
ω=2π/L
ε = H /L # (aspect ratios)
z_s = -x tan(alpha) # surface profile

Exp A	Exp B (FLOWLINE)	Exp C	Exp D (FLOWLINE)
alpha = 0.5 # deg	alpha = 0.5 # deg	alpha = 0.1 # deg	alpha = 0.1 # deg
beta2 -> inf	beta2 -> inf	beta2 =1000 + 1000 sin (ω x) · sin (ω y) # Pa a m^{-1}	beta2 =1000 + 1000 sin (ω x) Pa a m^{-1}
z_b = z_s − 1000 + 500 sin (ω x) sin (ω y)	z_b = z_s − 1000 + 500 sin (ω x)	z_b = z_s − 1000	z_b = z_s − 1000
Amplitude = 500 # m (perturbation amplitude)	Amplitude = 500 # m	Amplitude = 0 # m FLAT BED	Amplitude = 0 # m FLAT BED

2D Experiments (Flowline)

Experiment B: Ice flow over a rippled bed

- **Domain**: Flowline of length L (2D cross-section)
- **Bed topography**: `zb(x,y) = zs(x,y) - 1000 + 500·sin(ωx)`
- **Variation**: Bed elevation varies **only in x** direction
- **Mesh**: 1D flowline + vertical extrusion → **2D cross-sectional mesh**
- **Physics**: 2D flowline model (no lateral stresses)

Experiment D: Ice stream flow II

- **Domain**: Flowline of length L (2D cross-section)
- **Bed topography**: Flat
- **Friction**: `β²(x,y) = 1000 + 1000·sin(ωx)`
- **Variation**: Basal friction varies **only in x** direction
- **Mesh**: 1D flowline + vertical extrusion → **2D cross-sectional mesh**
- **Physics**: 2D flowline model (no lateral stresses)

(B & D) test:

- Longitudinal stress effects only
- Simpler flowline dynamics
- Computational efficiency for 1D models

3D Experiments (Full Domain)

Experiment A: Ice flow over a bumpy bed

- **Domain**: Square of side L (3D)
- **Bed topography**: `zb(x,y) = zs(x,y) - 1000 + 500·sin(ωx)·sin(ωy)`
- **Variation**: Bed elevation varies in **both x AND y** directions
- **Mesh**: 2D horizontal mesh + vertical extrusion → **3D volume mesh**
- **Physics**: Full 3D ice flow with lateral stress transmission

Experiment C: Ice stream flow I

- **Domain**: Square of side L (3D)

- **Bed topography**: Flat (`zb(x,y) = zs(x,y) - 1000`)
- **Friction**: `β²(x,y) = 1000 + 1000·sin(ωx)·sin(ωy)`
- **Variation**: Basal friction varies in **both x AND y** directions
- **Mesh**: 2D horizontal mesh + vertical extrusion → **3D volume mesh**
- **Physics**: Full 3D ice flow with lateral stress effects from friction patterns

(A & C) test:

- Lateral stress transmission
- 3D velocity fields
- Ice flow around obstacles (A) or between high/low friction zones (C)

===
=======

Geometry & mesh

The settings in dumb_ismip.py are currently hmax = 80 m and nvert = 10 which means that as the domain grows the computational cost becomes

L (km)	nx = L / 80 m	3-D elements ≈ nx² × (nvert–1)	RAM needed*
160	2 000	2 000² × 39 ≈ **156 M**	**>1 TB**
80	1 000	1 000² × 32 ≈ 32 M	200–300 GB
40	500	500² × 26 ≈ 6.5 M	40–60 GB
20	250	250² × 20 ≈ 1.2 M	8–10 GB
10	125	125² × 14 ≈ 0.22 M	~2 GB
5	63	63² × 10 ≈ 0.04 M	<1 GB

THIS IS NO GOOD for my replication of experiments
Pattyn's intercomparison recommended keeping horizontal spacing no larger than **~10–20 × Δz**.

is Δz = nvert???

What *is* `hmax` ?

*In BAMG/ISSM it is the **target maximum edge length** of the planar triangles (or quadrilaterals) that tessellate the x − yx−y footprint of glacier.*

Set `hmax` small → many small elements; set it large → few coarse elements.

From `hmax` → element count

For a square box of side L

elements in
x ≈ L *hmax*
y ≈ L hmax

Add the *vertical* layers (`nvert`) to extrude the 2-D mesh into prisms:
FOR A **UNIFORM STRUCTURED MESH** (LOWERBOUND FROM WHAT I ACTUALLY HAVE)
elem 3-D ≈ (L / hmax)^2 × (nvert−1).

Example
L=160 km,
hmax=80 m,
nvert=40

`elements per side L / hmax = 160 000 m / 80 m ≈ 2 000 3-D elements 2 000² × 40 ≈ 1.56 × 10⁸`

That is **156 million elements.** THAT'S RIDICULOUS

Elements → unknowns (degrees of freedom)

With a Full-Stokes model each **node** carries 4 primary unknowns

(u,v,w, p)(u,v,w,p).
A **linear prism** element in 3D has **6 nodes** (3 on top, 3 on bottom face).
6 nodes → 24 unknowns (**6 × 4 = 24 DOFs (degrees of freedom)**)
BUT not every DOF is unique per element, because **nodes are shared** between neighboring elements.
SO a more realistic depiction is
Total DOFs ≈ 2 × Number of elements

So the 156 M elements above yield **~300 M unknowns**.

Effect of loosening `hmax`

If you let `hmax` grow with the box length:

L (km)	`hmax` (m)	3-D elements	DOFs	Approx. RAM for A
160	**250**	40 M	80 M	~90 GB

L (km)	`hmax` (m)	3-D elements	DOFs	Approx. RAM for A
80	320	10 M	20 M	~22 GB
40	400	1.6 M	3 M	~3 GB
20	500	0.25 M	0.5 M	<1 GB
10	**80**	0.22 M	0.4 M	<1 GB
5	**80**	0.04 M	0.08 M	few hundred MB

These numbers drop by another ~60 % if you run the **2-D flowline** experiments (B & D) because the mesh is extruded only in one horizontal direction.

1. **`hmax` controls cost quadratically** in 3-D (1/hmax21/h_\text{max}^21/hmax2).
2. A universal 80 m spacing is fine for small domains but impossible for 160 km.
3. Matching Pattyn's guidance (Δx ≈ 10–20 Δz) lets you scale `hmax` up to 250–500 m for the long boxes without losing physical accuracy and keeps the problem under ~100 GB—doable on a modest cluster node.

Loosening `hmax` from 80 m to, say, 320 m for the 80 km domain, you cut element count by a factor of **16** and memory by a similar factor, turning an infeasible run into something you can actually solve overnight.

ASPECT RATIO RULES in flowband to match `dumb_ismip.py`

originally I had these as my rules based on `dumb_ismip.py`
The rule here is I increase hmax by 25% every domain increase.
then round hmax, geometry sampling and nvert
the rule for the geometry sample is simple I just calculate it to be ~1.6 times smaller than hmax and keep 80 as minimum
this proportionally matches every domain length parameters to those of the base case

domain length (km)	5	10	20	40	80	160
hmax	40	100	125	156	195	244
nvert	20	20	20	20	20	20
(nx) geometry sampling	40	80	80	97	122	152

code

```
def choose_mesh_spacings(L, exp):
    """
    Rules:
    - Base: 5km domain with hmax=80m, nvert=10, nx=50
    - hmax scaling: 25% for each doubling
    - TAR = hmax/nvert = 8
    - Geometry: spacing = hmax/1.6
    """
    if exp in ('B', 'D'):
        print(f"\n=== Flowband EXPERIMENT {exp} ===")
        # ===== base case: 5km =====
        base_L = 5000
        base_hmax = 80
        nx_base = 80
        nvert = 20

        if L <= base_L:
            # 5km or smaller
            hmax_xy = int(base_hmax/ 2)
            nx = int(nx_base / 2)

        else:
            # 25% for each increase in domain length
            doublings = np.log2(L / base_L) # example: np.log2(40000 / 
base_L) = 3.0
            hmax_xy = base_hmax * (1.25 ** doublings)
            # round
            hmax_xy = round(hmax_xy, 1) if hmax_xy >= 100 else 
round(hmax_xy)
            # ===== GEOMETRY SAMPLING =====
            geometry_spacing = hmax_xy / 1.6 #
            nx = int(max(nx_base, geometry_spacing))  # establish minimum as 
nx_base

    else:
        # ===== 3D CASES - COARSER RESOLUTION =====
        print(f"\n=== 3D EXPERIMENT {exp} ===")

        if L == 10000:
            hmax_xy = 300   # m

        elif L == 20000:
            hmax_xy = 600   # m

        elif L == 40000:
```

```python
            hmax_xy = 2000   # m

        elif L > 40000:
            # fall back for largest domain
            hmax_xy = 6000   # m

        else:
            # coarser base resolution for 3D
            hmax_xy = 150   # m

        nvert = 8   # vertical layers
        nx = int(L / hmax_xy) + 1  # geometry points

    # ===== COMMON CALCULATIONS =====
    elements_along_flow = L / hmax_xy
    aspect_ratio = hmax_xy / nvert
    L_km = L / 1000

    # ===== QUALITY CHECKS =====
    print(f"\nMesh for L = {L_km:.0f} km")
    print(f"   hmax_xy  = {hmax_xy:.0f} m")
    print(f"   nvert    = {nvert}")
    print(f"   nx       = {nx}")
    print(f"   Elements along-flow = {elements_along_flow:.0f}")

    return nvert, hmax_xy, nx
```

Velocities units check

Pattyn (2008) tables are in **m a⁻¹** and **kPa**.
If necessary, rescale by `yts` and `/1e3` respectively so numbers sit on the same axes as benchmark

Analysis

in `stressbalance.py` (line 184 onwards)

```python
 def marshall(self, prefix, md, fid):  # {{{

        WriteData(fid, prefix, 'object', self, 'class', 'stressbalance',
'fieldname', 'vertex_pairing', 'format', 'DoubleMat', 'mattype', 3)

        yts = md.constants.yts
```

```
        WriteData(fid, prefix, 'object', self, 'class', 'stressbalance',
'fieldname', 'spcvx', 'format', 'DoubleMat', 'mattype', 1, 'scale', 1. /
yts, 'timeserieslength', md.mesh.numberofvertices + 1, 'yts', yts)
        WriteData(fid, prefix, 'object', self, 'class', 'stressbalance',
'fieldname', 'spcvy', 'format', 'DoubleMat', 'mattype', 1, 'scale', 1. /
yts, 'timeserieslength', md.mesh.numberofvertices + 1, 'yts', yts)
        WriteData(fid, prefix, 'object', self, 'class', 'stressbalance',
'fieldname', 'spcvz', 'format', 'DoubleMat', 'mattype', 1, 'scale', 1. /
yts, 'timeserieslength', md.mesh.numberofvertices + 1, 'yts', yts)
```

shows the conversion process
My input velocities are in **m/year**

- `scale=1./yts` converts them to **m/s** when writing to the C++ solver
- `yts` = "years to seconds" conversion factor (≈ 31,557,600)
- Results are converted back to **m/year** when loaded (?)

> > > > LETS INVESTIGATE!

1. **`solve.py`** → calls `marshall()` and `loadresultsfromcluster()`
2. **`marshall.py`** → general wrapper, no specifics
3. **`loadresultsfromcluster.py`** → calls `loadresultsfromdisk()`
4. **`loadresultsfromdisk.py`** → calls `parseresultsfromdisk()`

```
find . -name "*parseresults*" -type f
```

In `parseresultsfromdisk.py` lines 106-117:
python

```
# Process units here FIXME: this should not be done here!
yts = md.constants.yts
...
elif fieldname == 'Vx':
    field = field * yts    # ← THE REVERSE CONVERSION!
elif fieldname == 'Vy':
    field = field * yts
```

Velocity range: 0.0 to 1639.1561693461979 is fast but reasonable in units m/yr

PERIODIC BCS in 3D CHECK

geometry has a **slope** (α = 0.5°), so the left and right boundaries have different absolute z-coordinates:

- **Left boundary**: z ranges from -1000 to 0 (surface at x=0)
- **Right boundary**: z ranges from -1043.6 to -43.6 (surface drops due to slope)

For ISMIP-HOM with sloping geometry, periodic boundary conditions need to match **relative positions within the ice column**, not absolute z-coordinates.

🔧 Geometry-Aware Matching

- **Old approach**: Match absolute (y,z) coordinates → Fails with slopes
- **New approach**: Match y-coordinates + **relative depth** within ice column

```
relative_depth = (vertex_z - bed_z) / thickness
```

- 0.0 = at the bed
- 1.0 = at the surface
- 0.5 = halfway through ice column

🔍 Why This Works

For ISMIP-HOM experiment A:

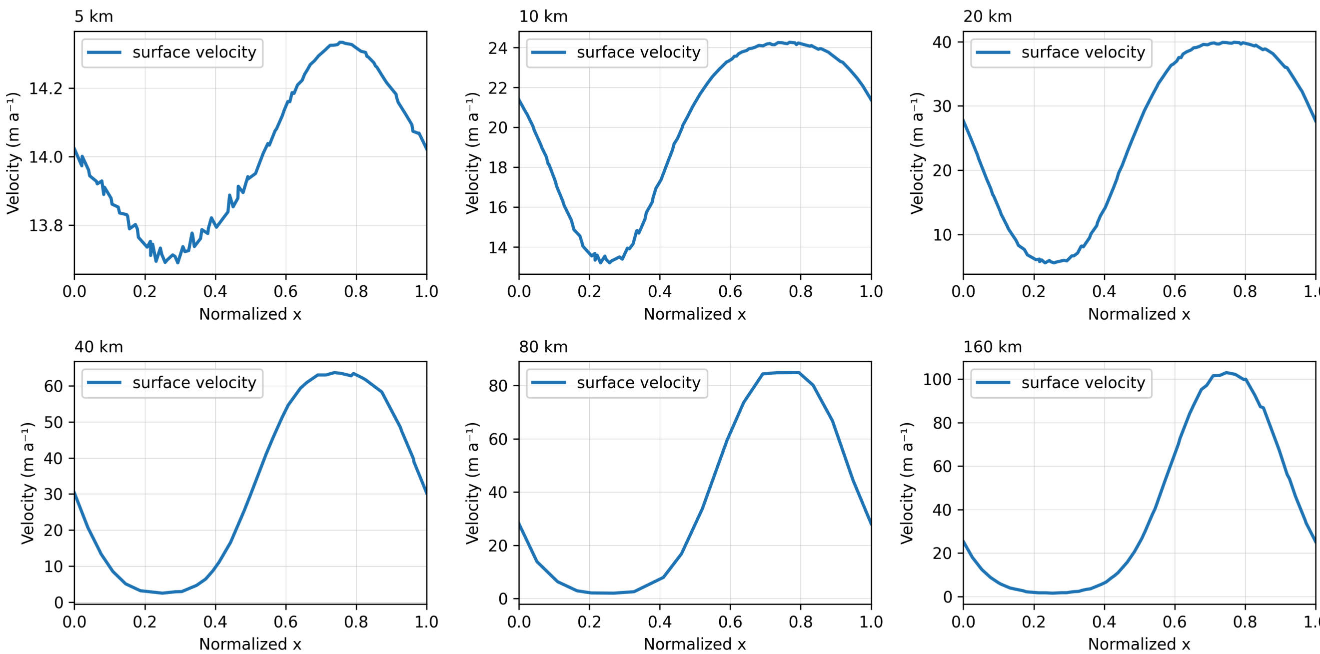
- Left boundary at x=0: surface elevation = 0
- Right boundary at x=5000: surface elevation = -43.6m (due to 0.5° slope)
- But a vertex at 50% depth should have the same physics on both sides!
- The new algorithm pairs vertices at the **same relative depth**, accounting for the geometric slope

EXP A and C size

domain length (km)	**5**	**10**	**20**	**40**	**80**	**160**
hmax	150	300	600	2000	4000	6000
nvert	8	8	8	8		8
nx (geometry sampling)	34	34	34	21		27
elements along flow	33	33	33	20		26
elements (2D)	2598	2598	2598	966		1652
vertices (2D)	1366	1366	1366	524		881

domain length (km)	5	10	20	40	80	160
elements (3D)	18186	18186	18186	6762		11564
vertices (3D)	10928	10928	10928	4192		7048

EXP A results



F. Pattyn et al.: Benchmarks for higher-order and full-Stokes ice sheet models 101

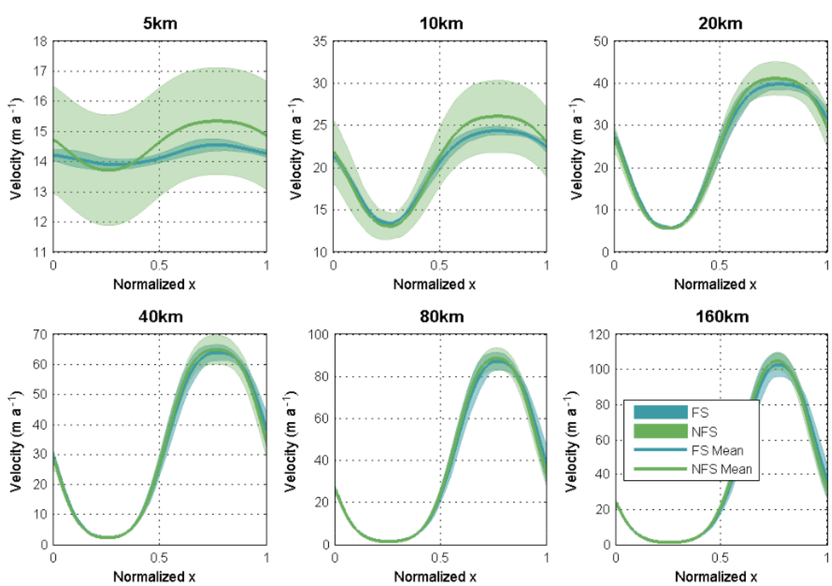


Fig. 5. Results for Exp. A: norm of the surface velocity across the bump at $y=L/4$ for different length scales L. The mean value and standard deviation are shown for both types of models.

5 km

Total elapsed time: 0 hrs 13 min 56 sec
Surface velocity ranges (m a^{-1}):
vx: [13.68, 14.43]
vy: [-2.55, 2.55]
vz: [-4.69, 4.45]

10 km

Total elapsed time: 0 hrs 11 min 46 sec
Surface velocity ranges (m a^{-1}):
vx: [13.20, 24.26]
vy: [-4.53, 4.55]
vz: [-6.68, 6.29]

20 km

Total elapsed time: 0 hrs 11 min 4 sec
Surface velocity ranges (m a^{-1}):
vx: [5.55, 39.92]
vy: [-5.50, 5.55]
vz: [-7.93, 7.44]

40 km

Total elapsed time: 0 hrs 2 min 21 sec
Surface velocity ranges (m a^{-1}):
vx: [2.50, 63.66]
vy: [-3.43, 3.58]
vz: [-8.18, 7.44]

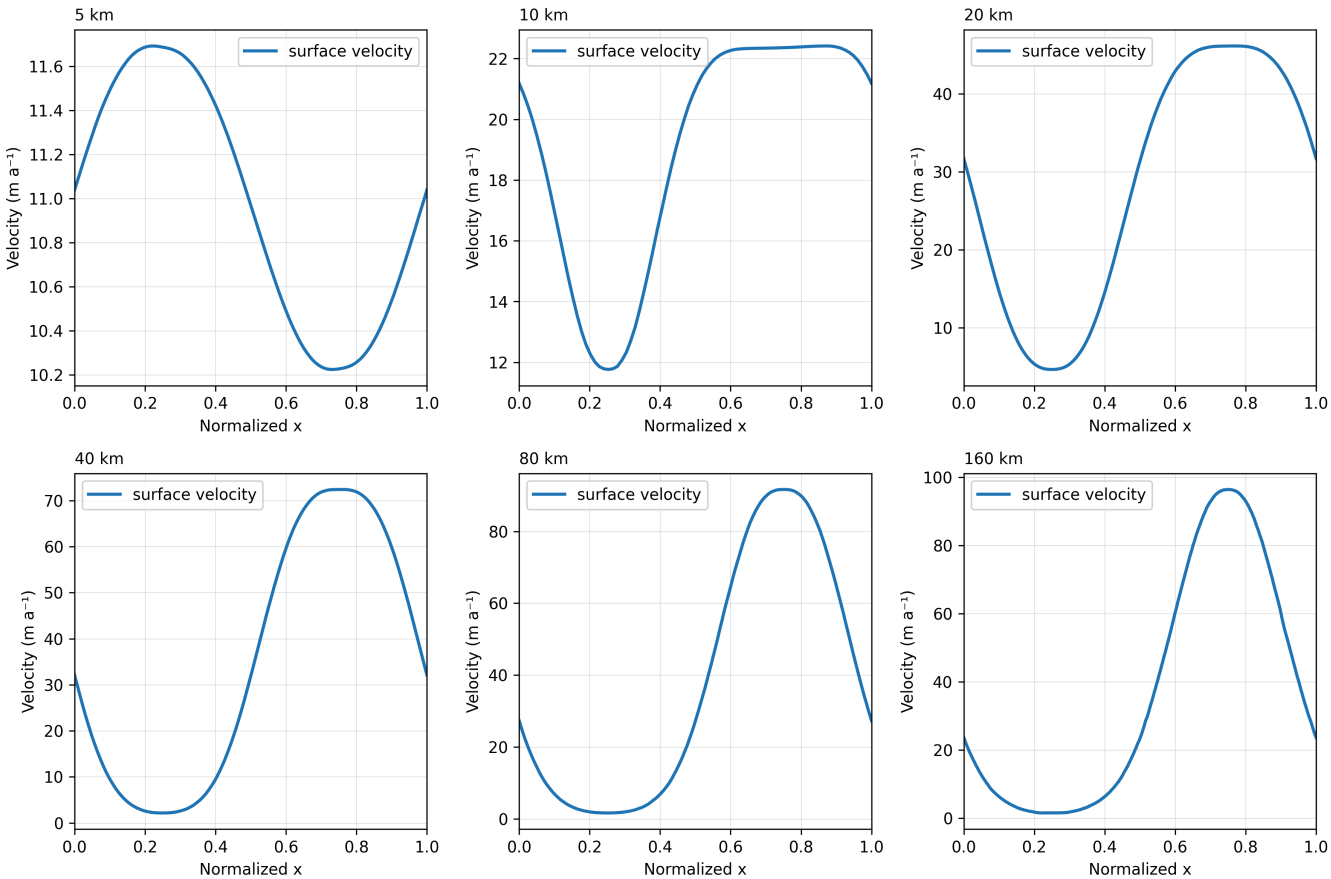
80 km

Total elapsed time: 0 hrs min sec
Surface velocity ranges (m a^{-1}):

160 km

Total elapsed time: 0 hrs 4 min 46 sec
Surface velocity ranges (m a^{-1}):
vx: [1.57, 103.09]
vy: [-0.77, 0.78]
vz: [-4.48, 3.13]

EXP B results



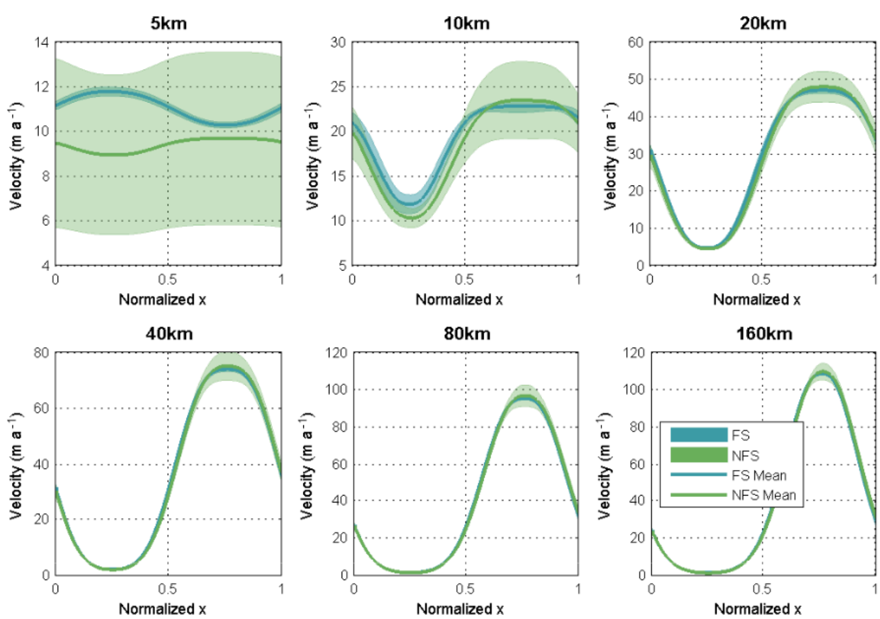
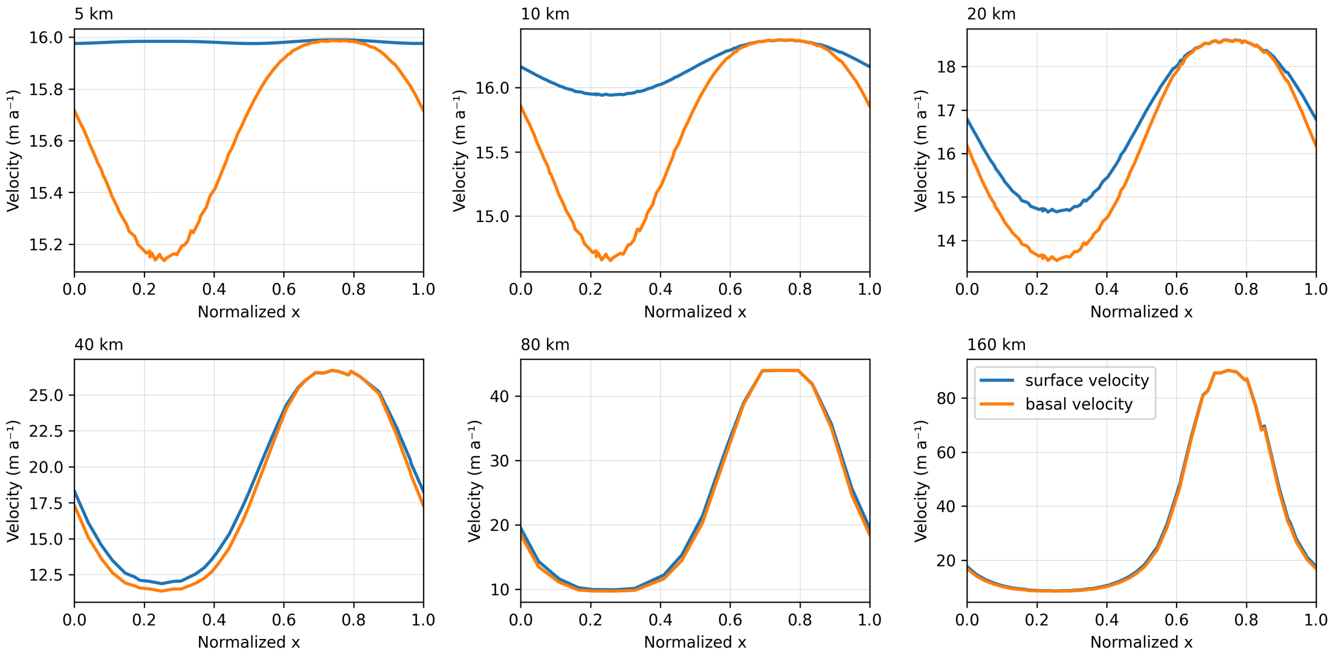


Fig. 6. Results for Exp. B: norm of the surface velocity for different length scales L. The mean value and standard deviation are shown for both types of models.

EXP C results



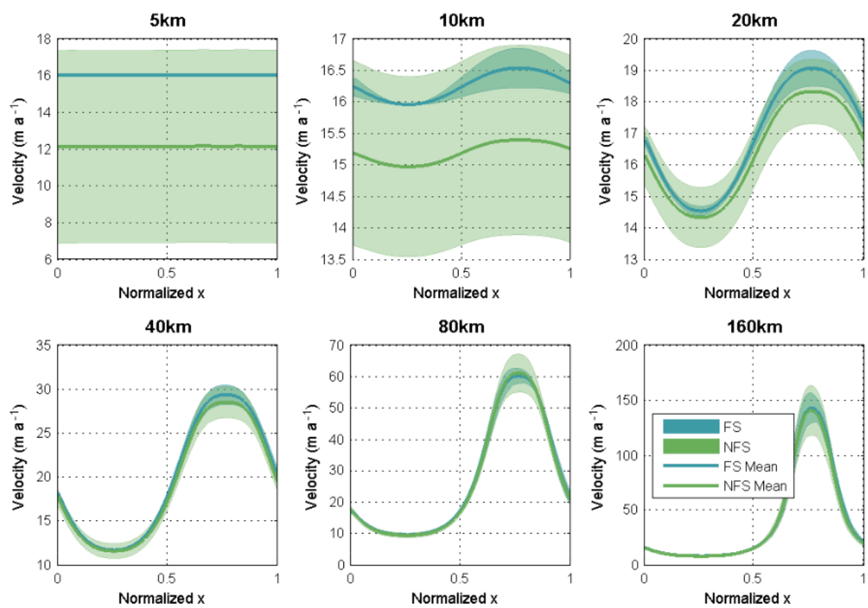
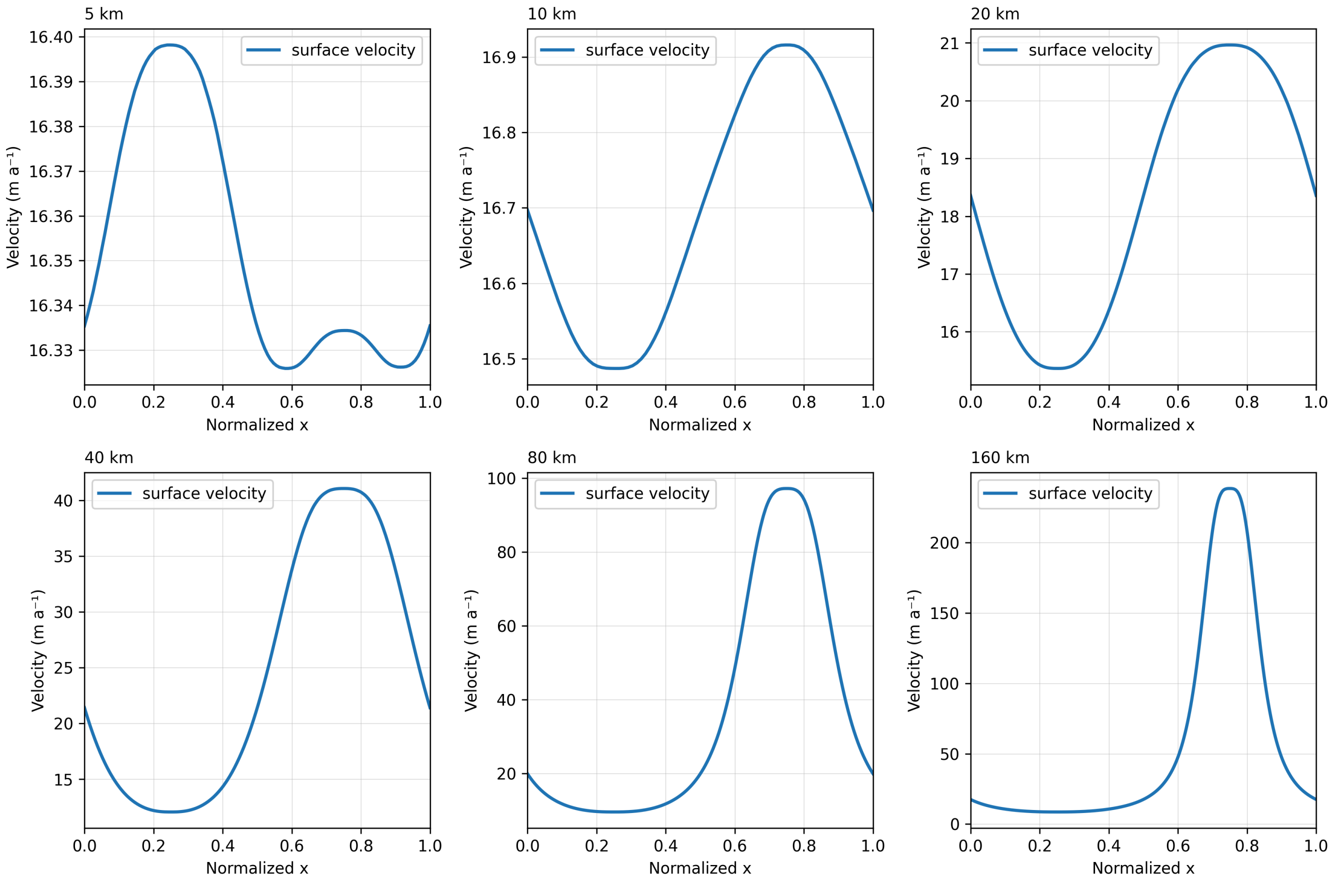


Fig. 8. Results for Exp. C: norm of the surface velocity at $y=L/4$ for different length scales L. The mean value and standard deviation are shown for both types of models.

EXP D



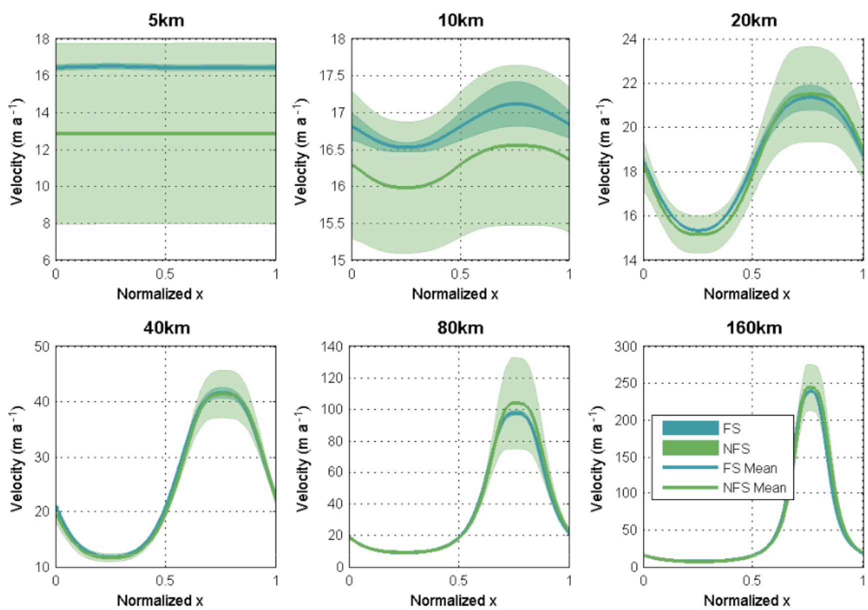


Fig. 9. Results for Exp. D: norm of the surface velocity for different length scales L. The mean value and standard deviation are shown for both types of models.

OTHER boundary conditions

- Constant SMB `0.5 m a⁻¹` introduces thickening that Pattyn doesn't include. Set it to zero unless you deliberately want a transient.

```
    # mass balance
   md.smb.mass_balance = 0.5 * np.ones(nv) # PATTYN DOES NOT INCLUDE THIS
BUT I ALREADY DEACTIVATED ISSMB
```

interfaces

surface	vertex flag	element flag	comment
basal interface	`md.mesh.vertexonbase` (called `vertexonbed` in very old versions) (issm.ess.uci.edu, issm.jpl.nasa.gov)	`md.mesh.elementonbed`	where you apply basal traction/friction
upper surface	`md.mesh.vertexonsurface`	`md.mesh.elementonsurface`	where you normally impose zero normal stress
any boundary (including side walls)	`md.mesh.vertexonboundary`	—	convenience mask; true on base, surface **and** lateral faces