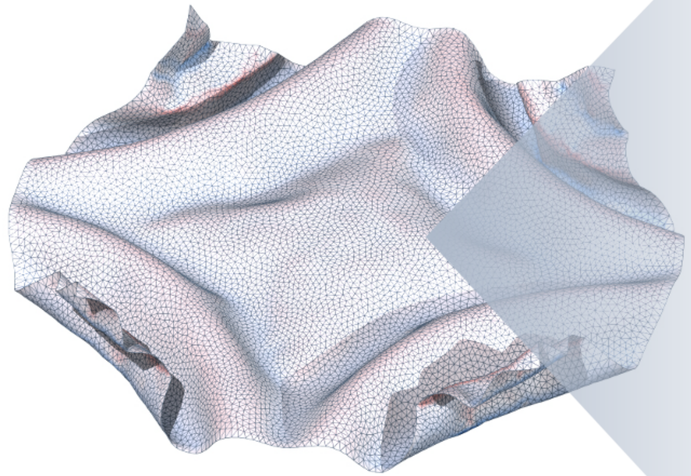
Computational models of elastic sheets III

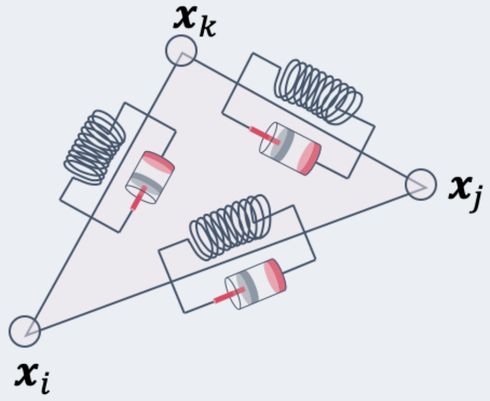
2026 Boulder Summer School for Condensed Matter and Materials Physics
June 6–10, 2026

Chris H. Rycroft, University of Wisconsin–Madison
(rycroft@wisc.edu)

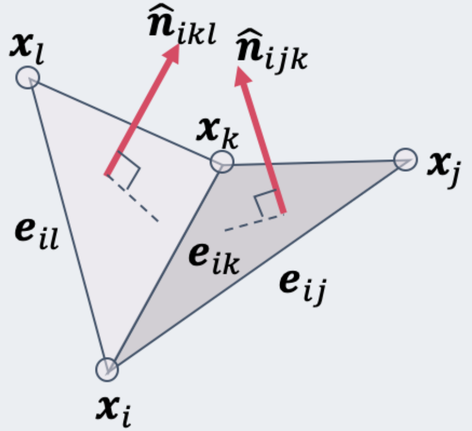
Computational model

Triangular mesh





in-plane interactions



out-of-plane rigidity

$$\mathbf{F}_i = -\sum_{j\in N_i} \nabla_{\mathbf{x}_i} E_s(\mathbf{r}_{ij}) + \sum_{j\in N_i} \mathbf{F}_d(\mathbf{r}_{ij}) - \sum_{(j,k,l)\in T_i} \nabla_{\mathbf{x}_i} E_b(\mathbf{n}_{ijk}, \mathbf{n}_{ikl}) + \sum_{\substack{j=0 \\ j\neq i, j\notin N_i}}^{n} \mathbf{F}_r(\mathbf{r}_{ij}) + \mathbf{F}_e(\mathbf{x}_i)$$

stretching | damping | bending | self-avoidance | external forces

$$\dot{\mathbf{x}}_i = \mathbf{v}_i$$
$$\dot{\mathbf{v}}_i = \mathbf{F}_i$$

ODE system
($6n$ components)

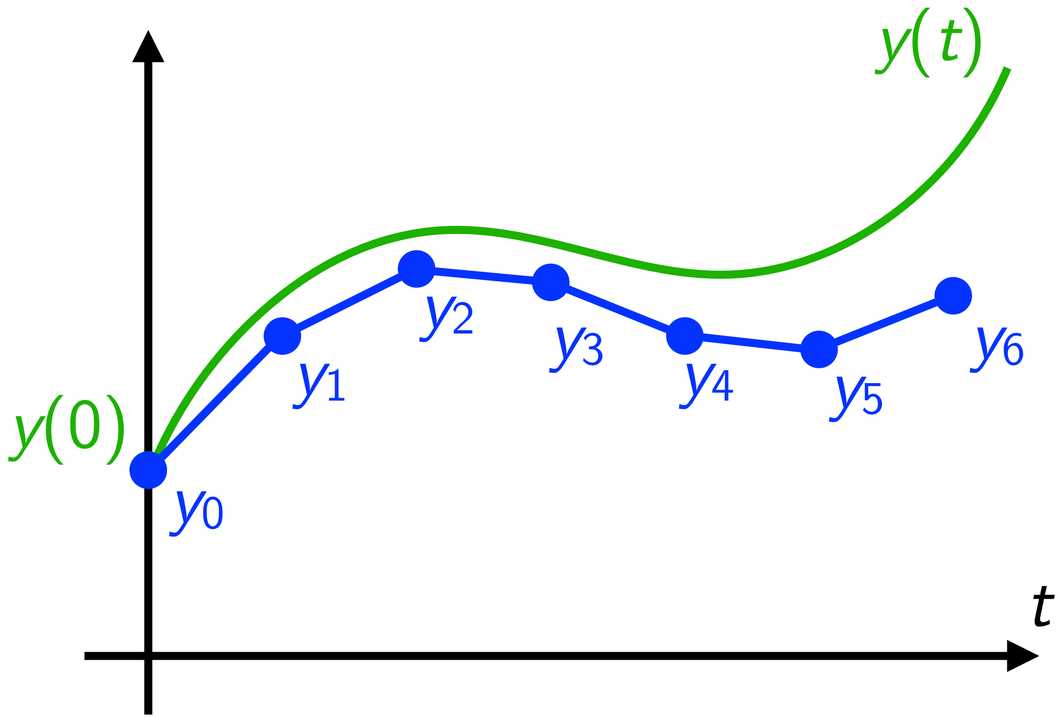
Integrating ODEs

- We are interested in numerically solving the ODE initial value problem

$$y' = f(t, y), \qquad y(0) = y_0$$

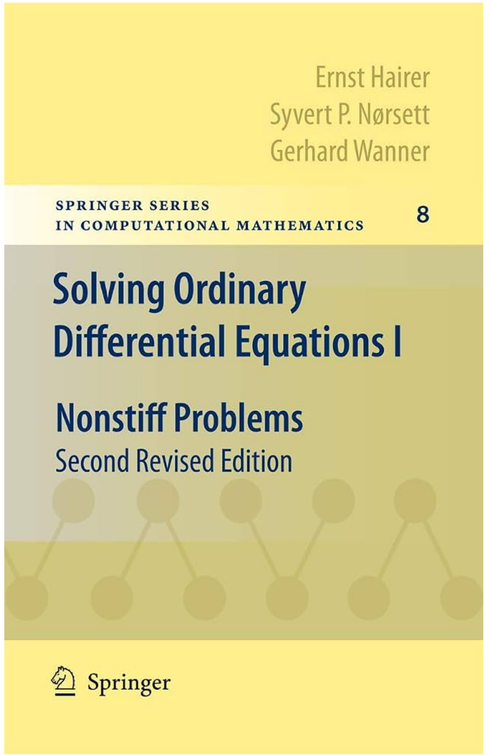
  where $y(t) \in \mathbb{R}^n$ and $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$

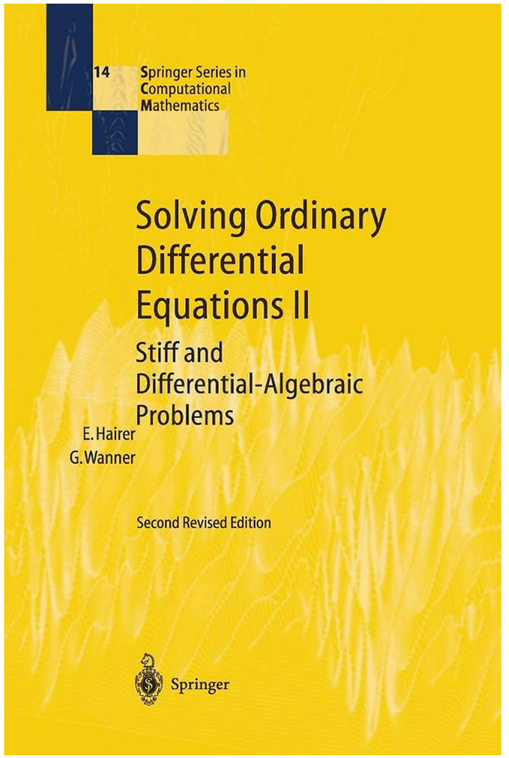
- Use an integration step size of h
- Let y_k be the numerical approximation of $y(t_k)$ where $t_k = hk$
- Set y_0 to be equal to $y(0)$



References

- For further information, see
  - Harvard AM205 website & notes
  - Harvard AM225 website & notes
  - Harvard AM205 YouTube videos
- See the following two textbooks
  - E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems,* Springer, 1993.
  - E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential–Algebraic Problems*, Springer, 1996.





"If anything has been made foolproof, then a better fool will be developed" (pp. 169, vol. I)

A first approach: the forward Euler step

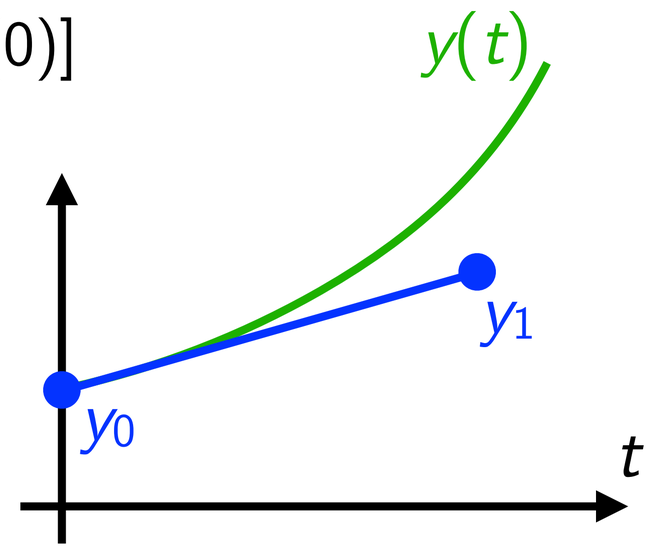
- Consider the numerical formula

$$y_{k+1} = y_k + hf(t_k, y_k)$$

- Compute difference between exact solution $y(t_1)$ and the numerical solution y_1:

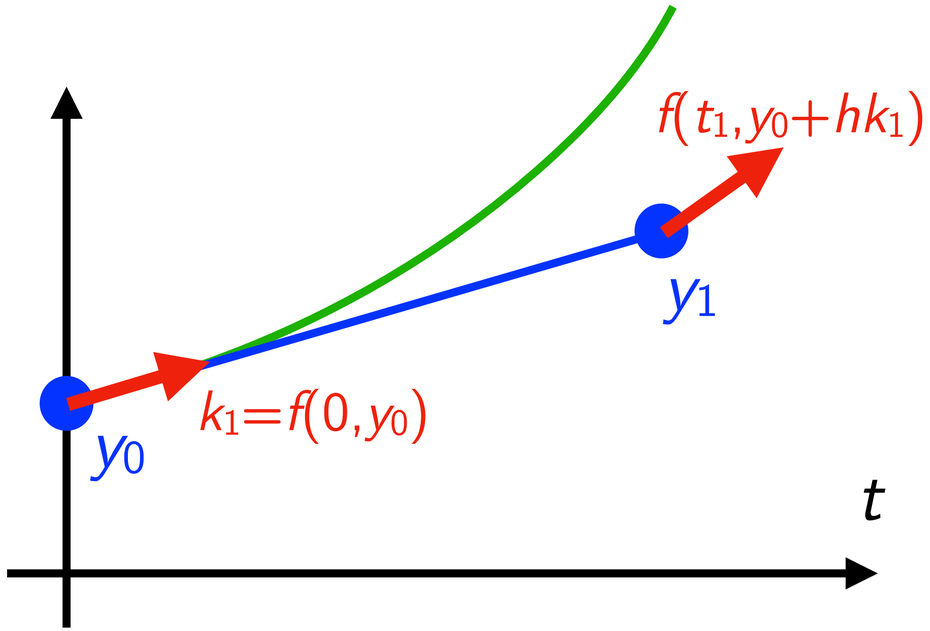
$$\begin{aligned} y(h) - y_1 &= y(h) - [y_0 + hf(h, y_0)] \\ &= \left(y(0) + hy'(0) + \frac{h^2}{2} y''(0) + O(h^3) \right) - [y(0) + hy'(0)] \\ &= \frac{h^2}{2} y''(0) + O(h^3) \end{aligned}$$

- Agrees up to powers of h: this a first-order method



Gaining more accuracy

- Euler method misses the curve of the exact solution
- But once we have computed y_1, then we could evaluate f there to estimate the slope at the end of the timestep

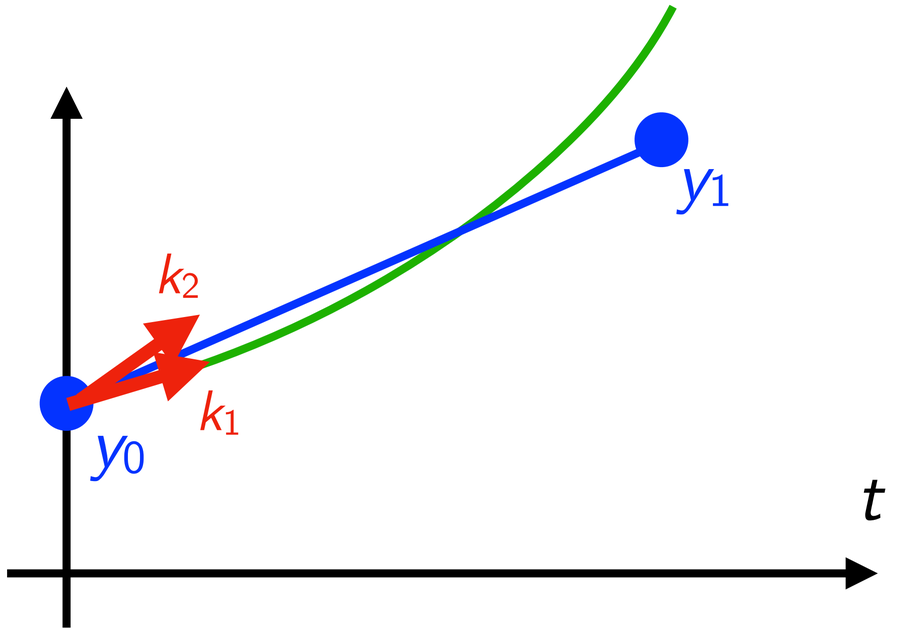


* In ODE literature k is used a subscript and also to signify intermediate stages. This is unfortunate.

Gaining more accuracy

- Euler method misses the curve of the exact solution
- But once we have computed y_1, then we could evaluate f there to estimate the slope at the end of the timestep
- Could combine both f evaluations to obtain better accuracy. The improved Euler method is

$$k_1 = f(t_k, y_k)$$
$$k_2 = f(t_{k+1}, y_k + hk_1)$$
$$y_{k+1} = y_k + \frac{h}{2}(k_1 + k_2)$$



Can show that $y_1 - y(h) = O(h^3)$

This is a second-order method

* In ODE literature k is used a subscript and also to signify intermediate stages. This is unfortunate.

The 'classical' fourth-order Runge–Kutta method

- These two are examples of Runge–Kutta (RK) methods that use multiple function evaluations to improve accuracy. The most famous RK method is

$$
\begin{aligned}
k_1 &= f(t_k, y_k) \\
k_2 &= f\left(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}k_1\right) \\
k_3 &= f\left(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}k_2\right) \\
k_4 &= f(t_k + h, y_k + hk_3) \\
y_{k+1} &= y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

- Four intermediate stages and fourth-order accurate
- Runge–Kutta methods refer to the broad class of methods like this. But the above method is so famous it is sometimes called "***the*** Runge–Kutta method", or the "classical RK4 method"

The Butcher tableau

- A general Runge–Kutta method from y_0 to y_1 has intermediate stages

$$k_i = f\left(x_0 + c_i h, y_0 + h \sum_{j=1}^{i-1} a_{ij} k_j\right)$$

- Solution is

$$y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i$$

- Method can be summarized in a Butcher tableau:

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} \\ \hline & b_1 & b_2 & \dots & b_{s-1} \quad b_s \end{array}$$

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Forward Euler

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

Improved Euler

$$\begin{array}{c|ccccc} 0 & & & & & \\ 1/2 & 1/2 & & & & \\ 1/2 & 0 & 1/2 & & & \\ 1 & 0 & 0 & 1 & & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Classical RK4

A test problem: the Brusselator

- To test the methods introduced so far, we will look at a two-component ODE system for $(y_1(t), y_2(t))$,

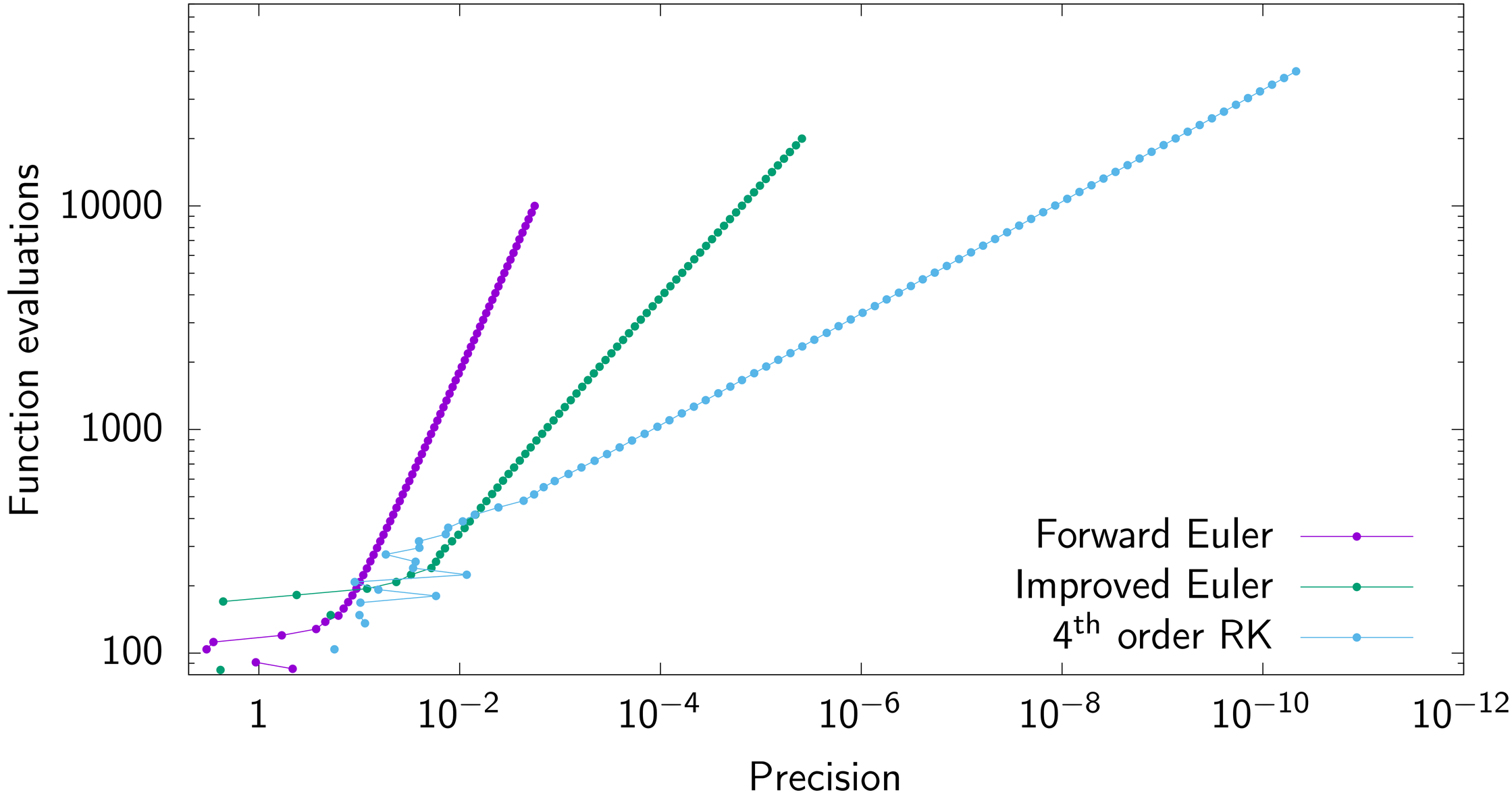
$$y_1' = 1 + y_1^2 y_2 - 4y_1,$$
$$y_2' = 3y_1 - y_1^2 y_2$$

- We use the initial condition

$$y_1(0) = 1.5, \qquad y_2(0) = 3$$

- This is a simple model of chemical kinetics. It is a good test since the smoothness varies over time.

Evaluating methods: the work–precision plot



Deriving RK methods: a simplification

- Our general ODE system is

$$y' = f(t, y), \qquad y(0) = y_0$$

  for $y(t) \in \mathbb{R}^n$ and $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$

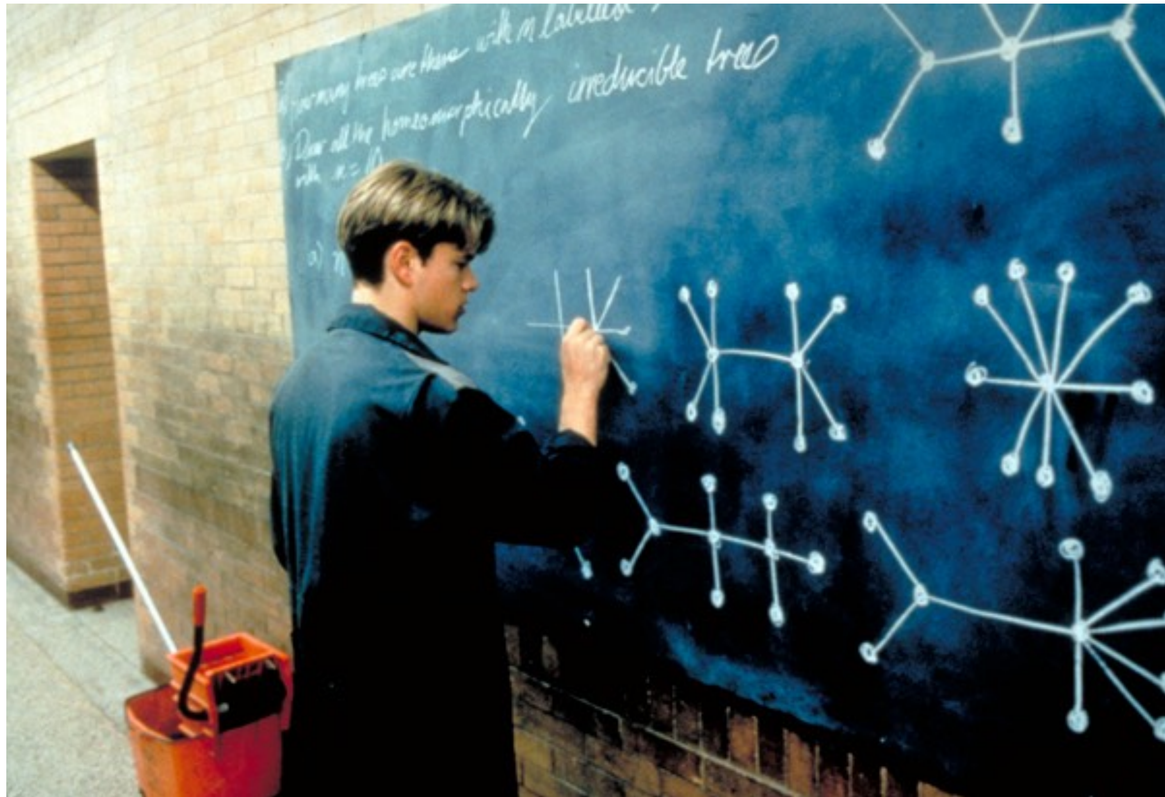
- To derive RK methods it is useful to consider the autonomous form where $T(t) = t$ is treated as another variable

$$\begin{pmatrix} T \\ y \end{pmatrix}' = \begin{pmatrix} 1 \\ f(T, y) \end{pmatrix}, \qquad \begin{pmatrix} T(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} 0 \\ y_0 \end{pmatrix}$$

- Writing $Y(t) = (T(t), y(t))$ where $Y(t) \in \mathbb{R}^{n+1}$ gives a simplified general form with explicit time dependence,

$$Y' = F(Y)$$

Blackboard interlude: the order conditions



Good Will Hunting,
Miramax Films, 1997.

Enumerating trees

- The number of trees grows quickly with the order p

Order	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
# trees	1	1	2	4	9	20	48	115	286	719
# conditions	1	2	4	8	17	37	85	200	486	1205

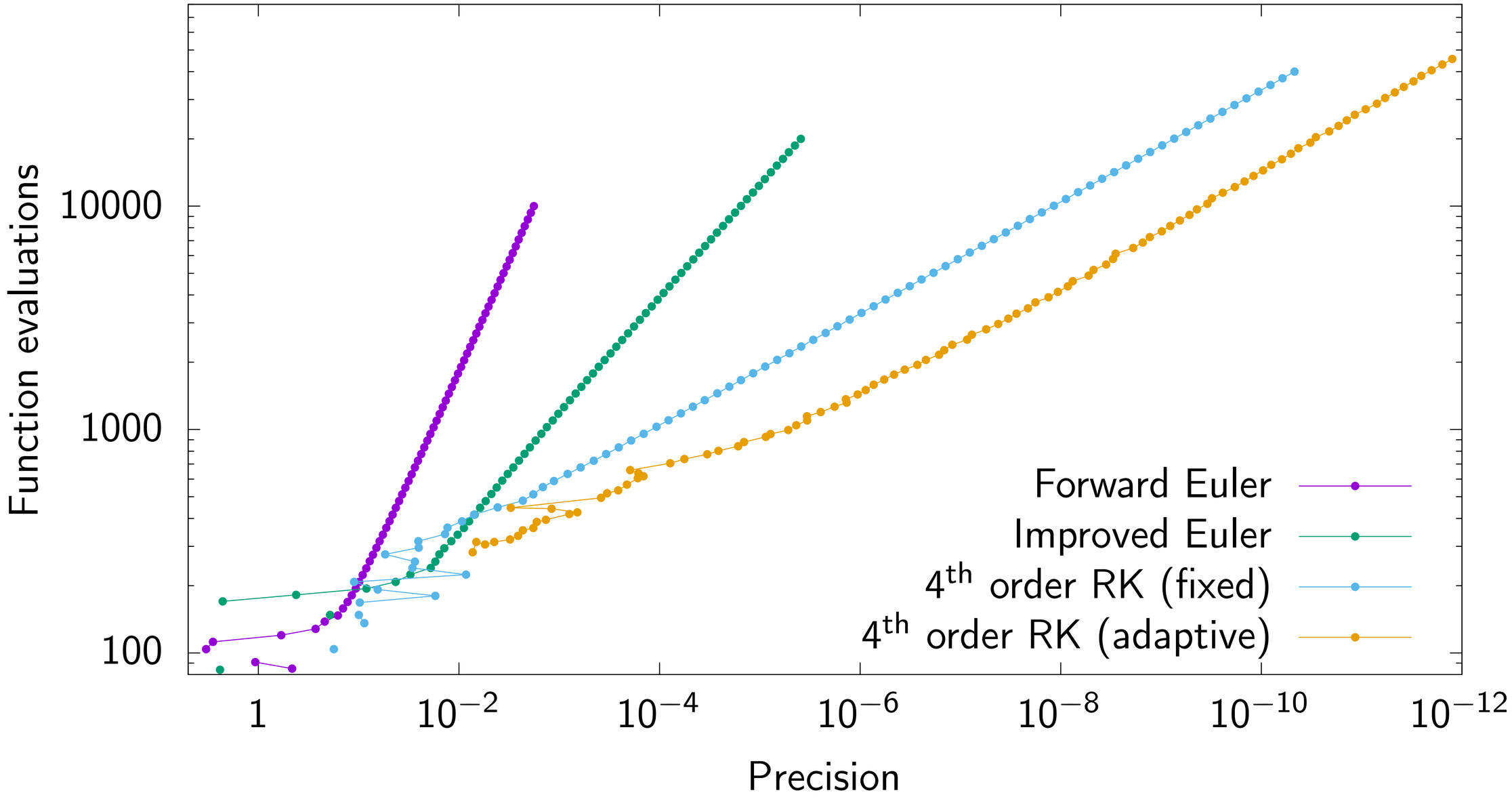
- Deriving higher-order methods becomes increasingly difficult
- For $p \geq 5$ it is not possible to derive a method with $s = p$ stages
- We must use $s > p$ stages to reach higher orders

Adaptive step size control

- Many examples (such as the sheet simulation) have varying smoothness
- We could take larger timesteps in regions where the solution is smoother
- Need to estimate error to do this. Derive methods that give two solutions y_1 and $\hat{y}_1$ of order p and q respectively
- Use $y_1 - \hat{y}_1$ as an estimate of error, and can control timestep to ensure it stays small enough (see derivation)

0					
$1/3$	$1/3$				
$2/3$	$-1/3$	1			
1	1	-1	1		
1	$1/8$	$3/8$	$3/8$	$1/8$	
y_1	$1/8$	$3/8$	$3/8$	$1/8$	
$\hat{y}_1$	$1/12$	$1/2$	$1/4$	0	$1/6$

Updated work–precision plot



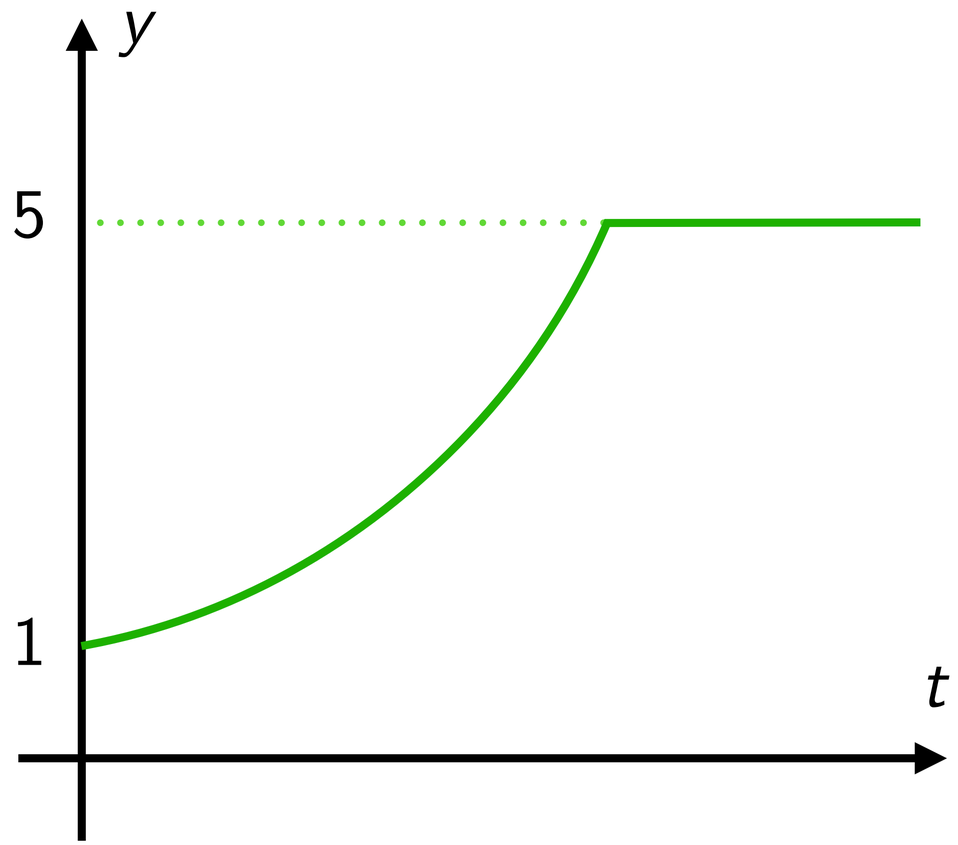
Additional benefit: handling discontinuities

- A key benefit of adaptive methods is that they can handle abrupt changes in an ODE. For example, consider the 1D ODE

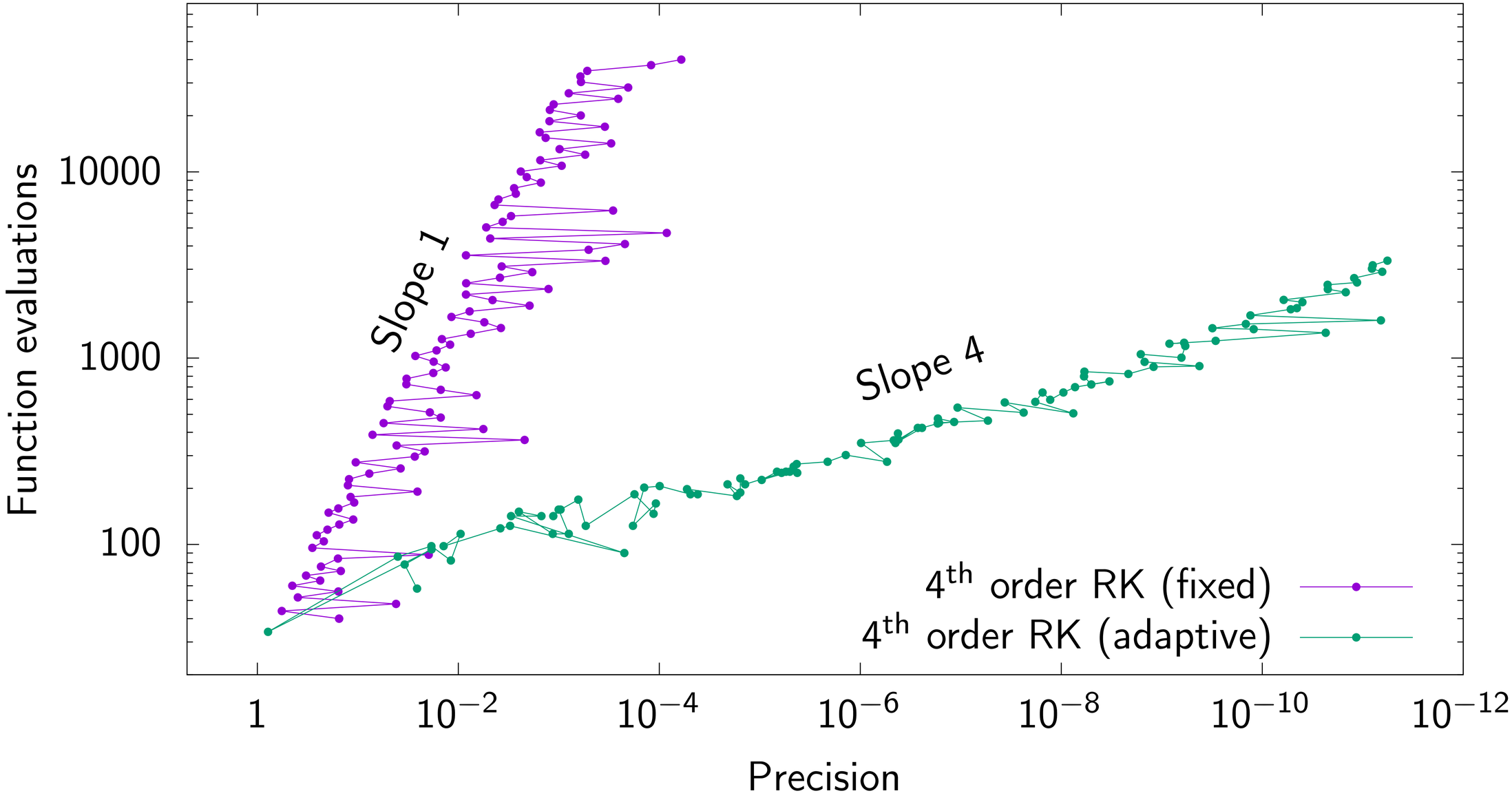
$$y' = \begin{cases} y & \text{if } y < 5, \\ 0 & \text{if } y \geq 5 \end{cases}$$

  with $y(0) = 1$

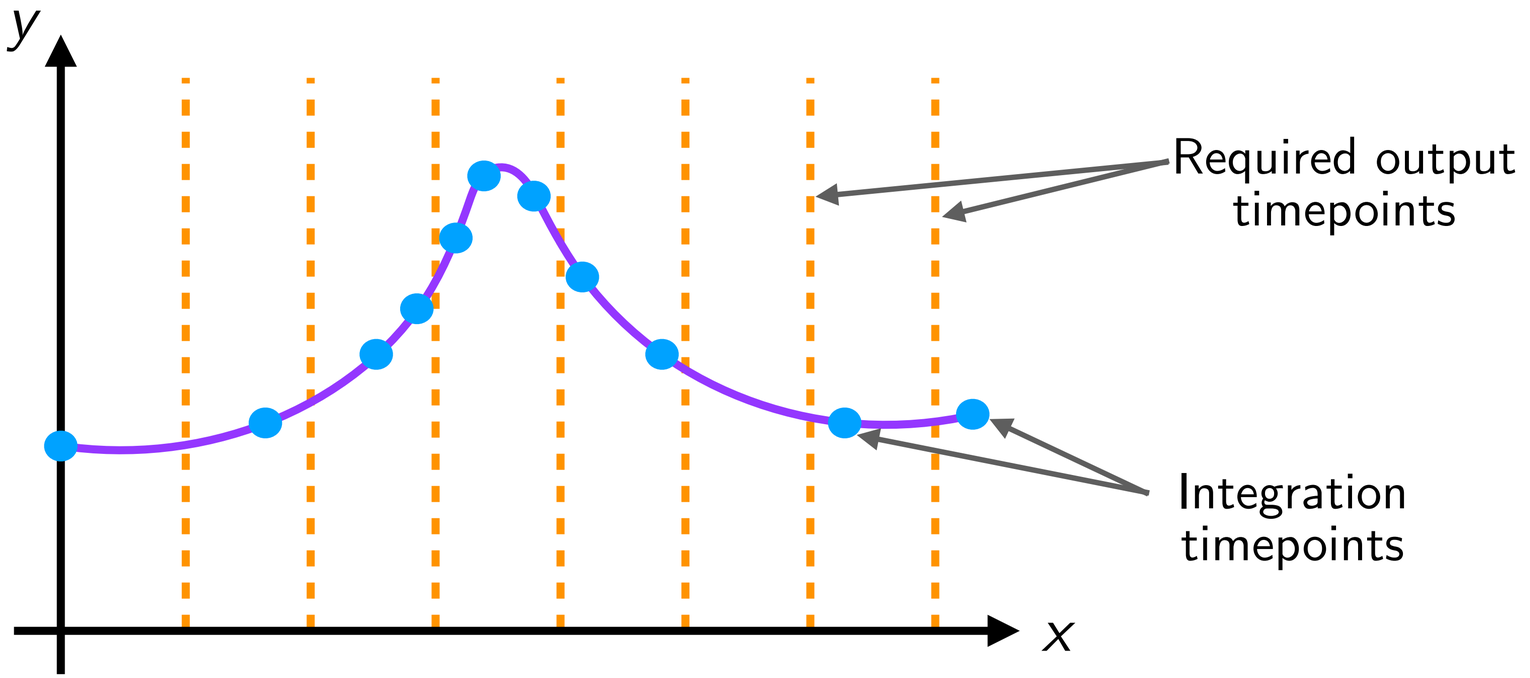
- How will the fixed-step and adaptive RK4 methods do on this case?



Comparison of fixed and adaptive timestepping

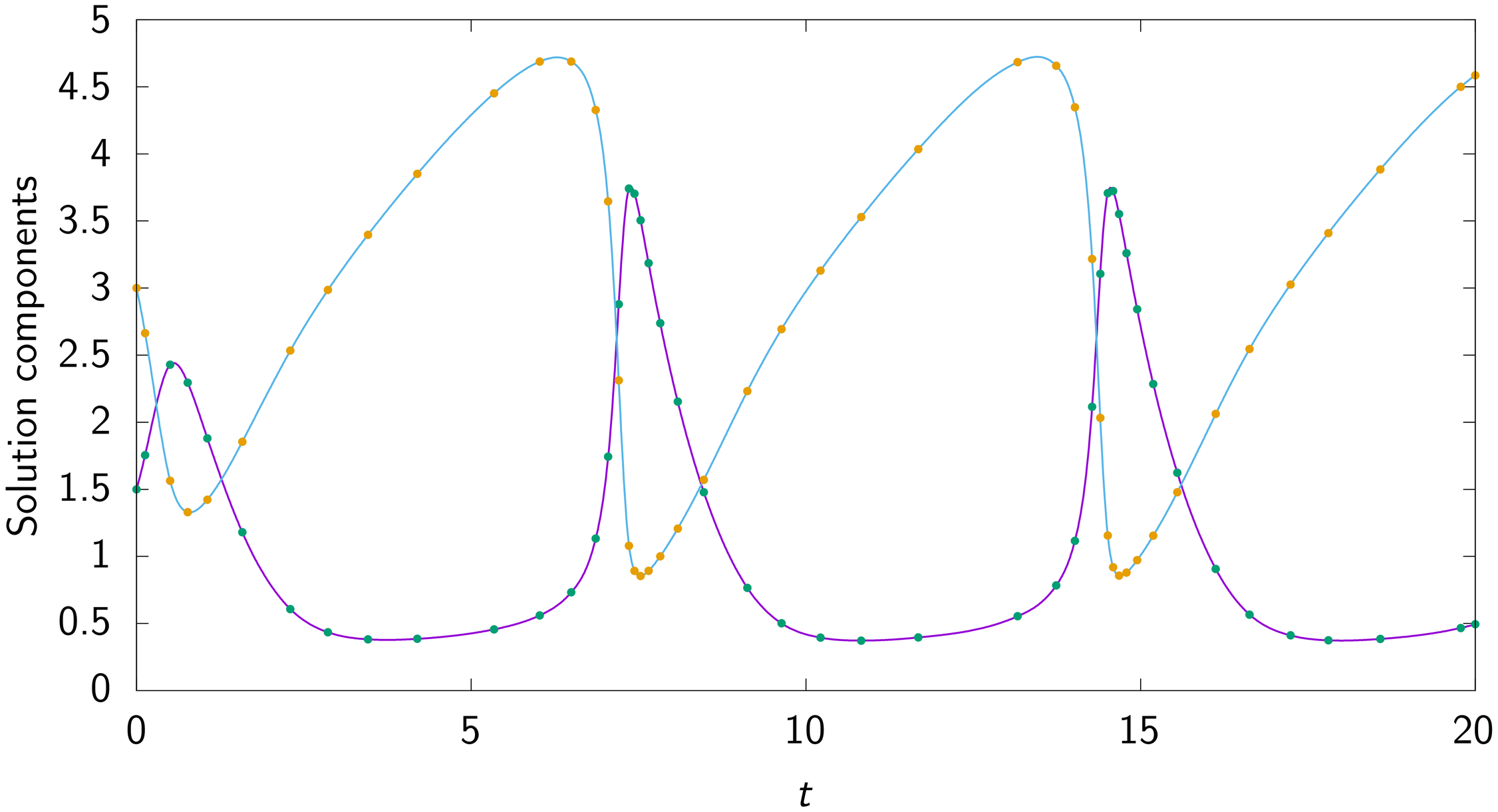


Dense output

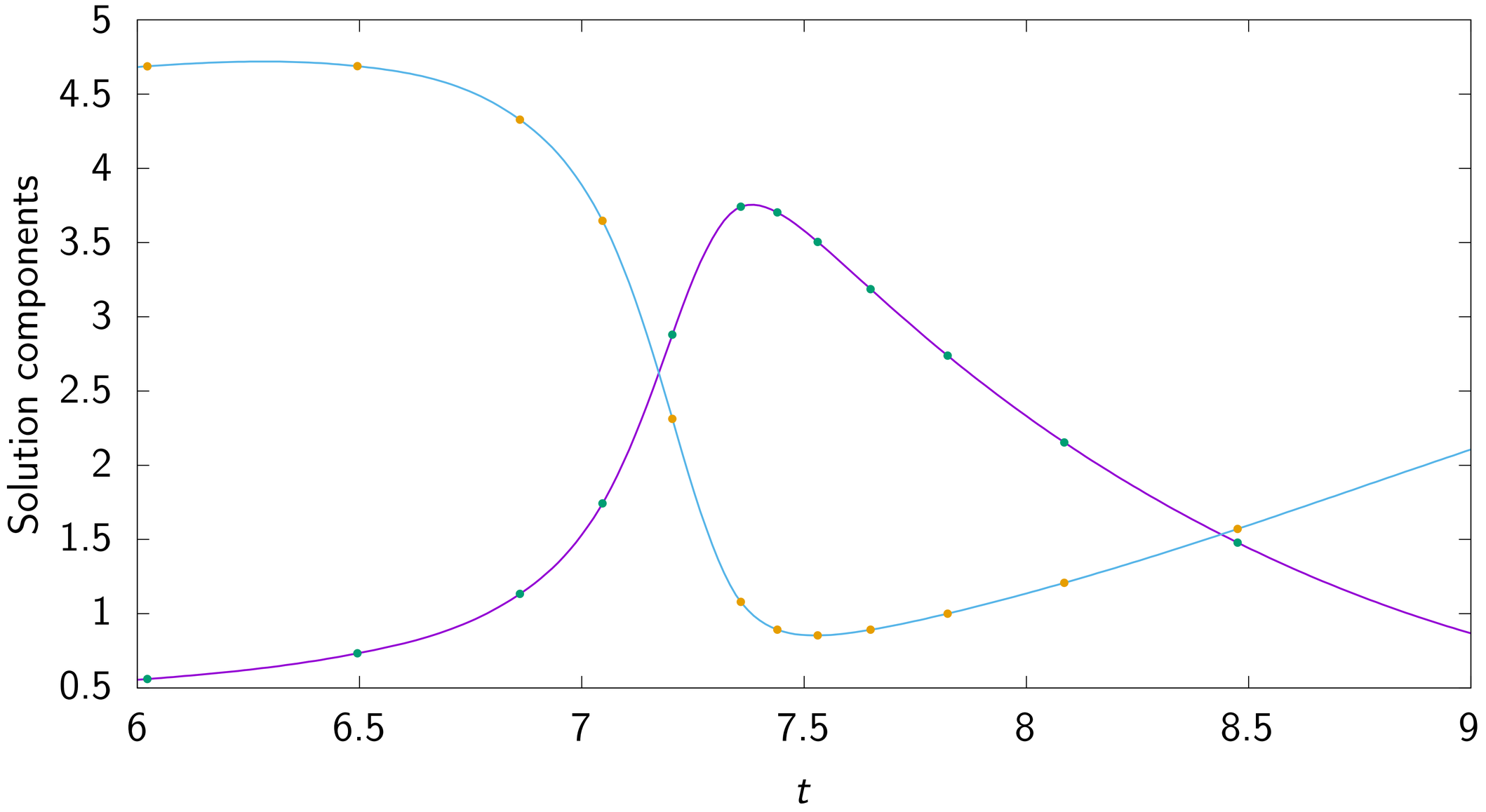


- Adaptive high-order RK methods require infrequent, intermittent timesteps. But often we need to output the solution at frequent, specific times.
- Simple solution: decrease timestep to exactly match the output times. Requires more timesteps and function evaluations.
- Better solution: create polynomial interpolant of order p^* over each timestep, then cheaply evaluate the solution over the entire interval.

Dense output for the Brusselator



Dense output for the Brusselator



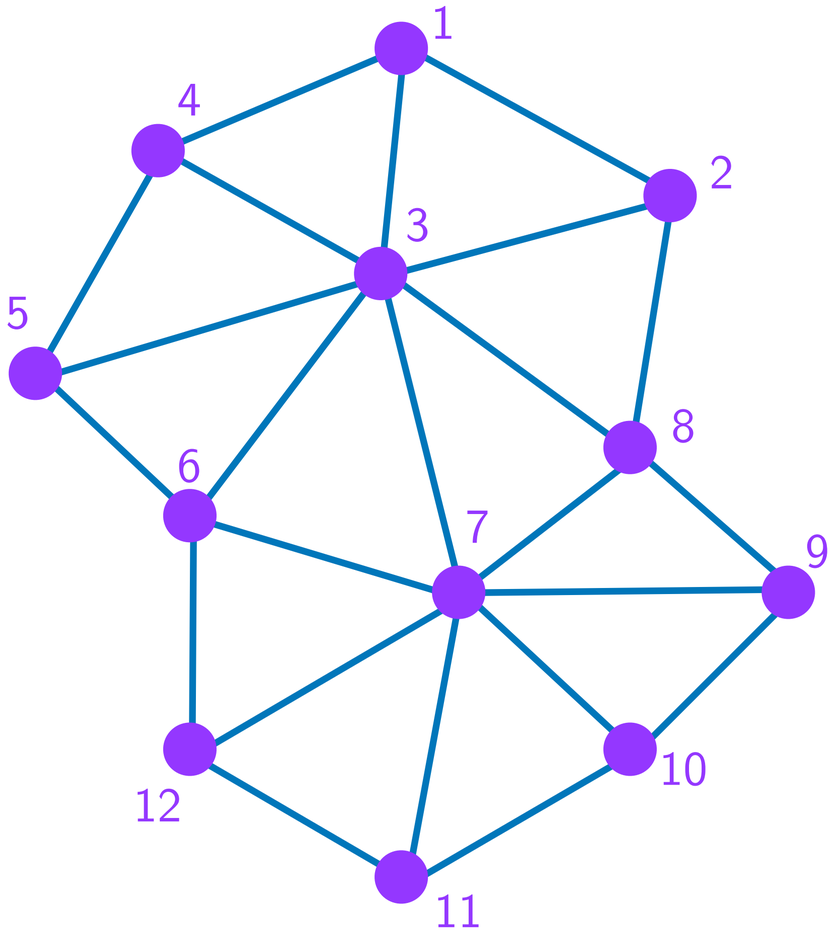
Sheet simulation: mesh topology

- From mesh generation, we build a representation of all the vertices and edges
- Each vertex has a list of all neighboring IDs, stored with a right-hand ordering
- For example for 7 in the interior, store

  3, 6, 12, 11, 10, 9, 8

- Boundary nodes are flagged to only contain a partial loop of neighbors, *e.g.* for 5 store

  6, 3, 4

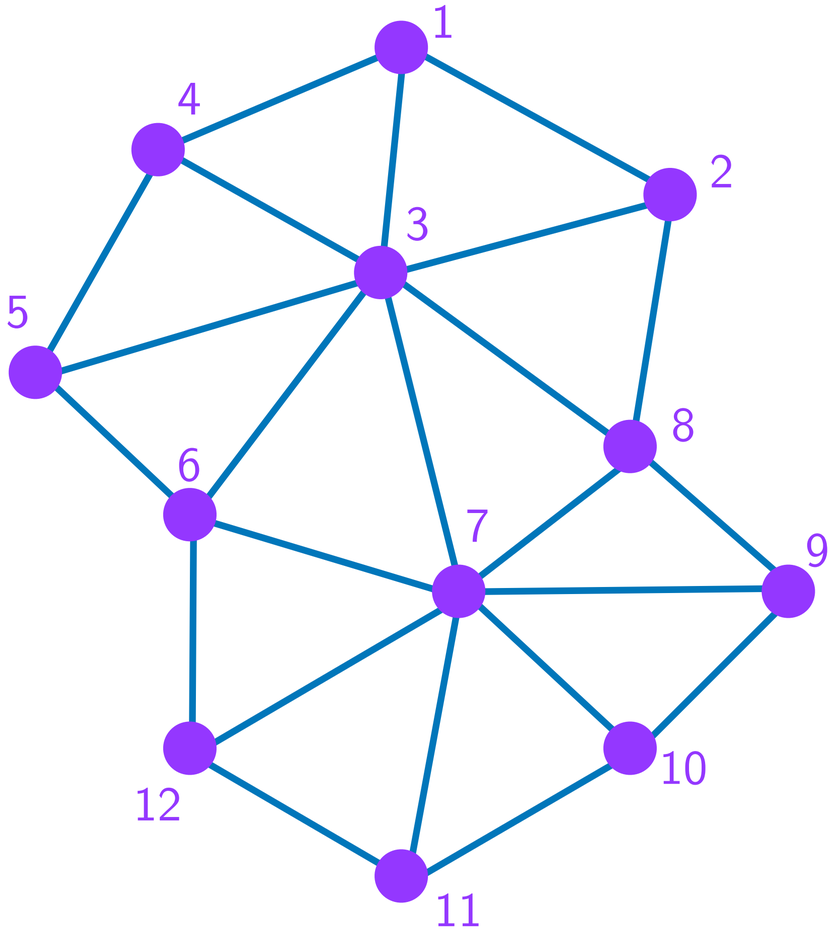


Sheet simulation: mesh topology

- At start of the simulation, we scan this information to build a set of ordered edge pairs (i,j) where $i < j$
- For example

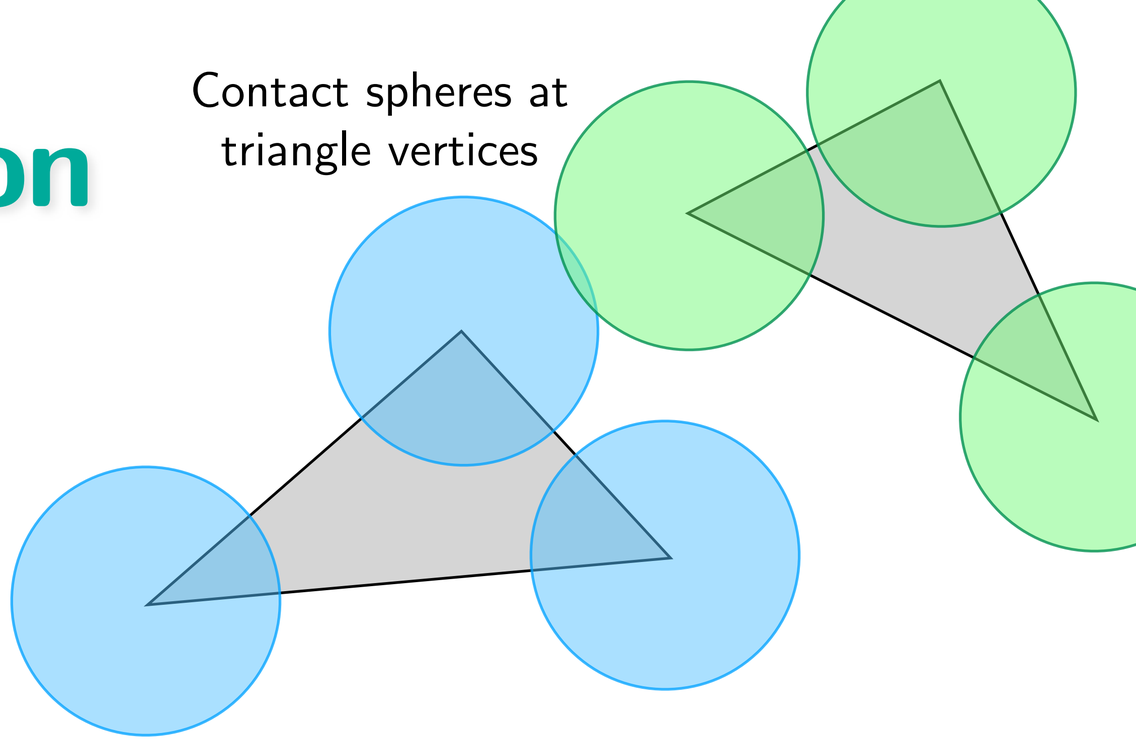
  (1,2), (1,3), (1,4), (2,8), (2,3), (3,4), (3,5), ...

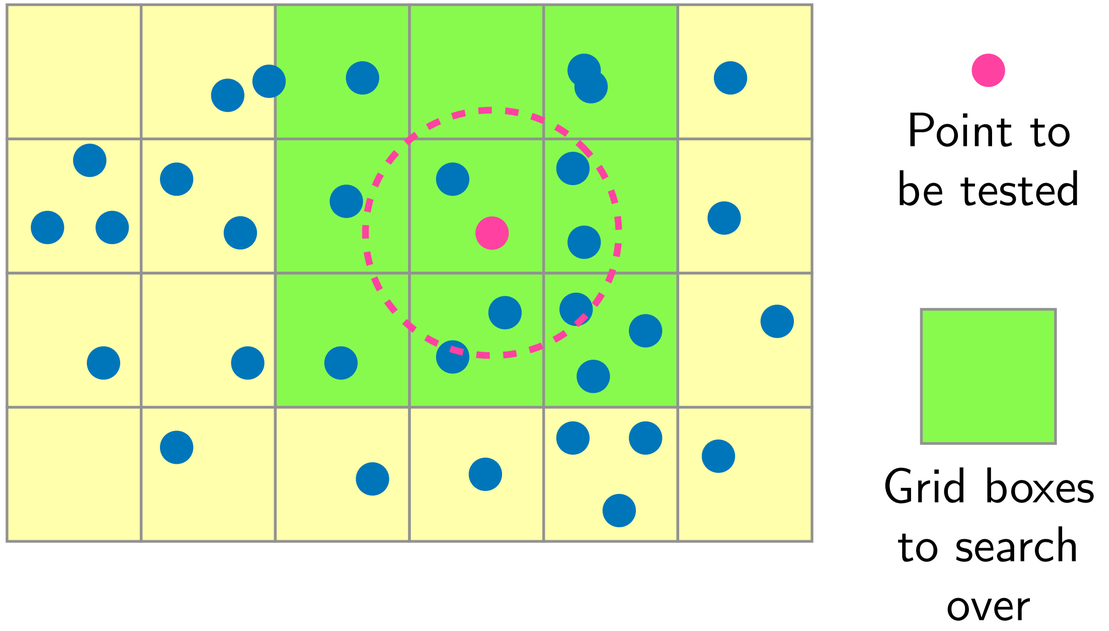
- Only need to consider each pair once to evaluate edge-based forces (via Newton's third law)
- Also build a table of oriented triangles (i,j,k) where $i<j$ and $i<k$



Efficient contact detection

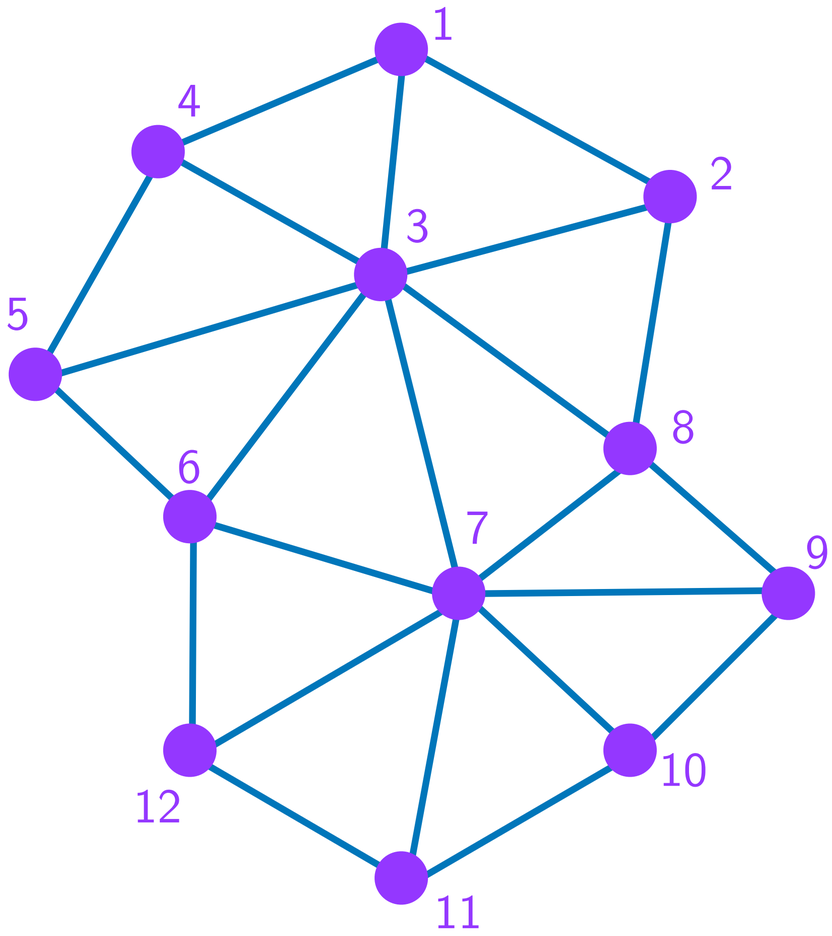
- During the simulation, it is necessary to find all overlapping pairs of contact spheres in order to evaluate self-avoidance
- In principle, it could be an $O(n^2)$ computation for n spheres
- Sort spheres into a spatial grid structure
- Only necessary to search spheres in nearby grid blocks, reducing complexity to $O(n)$





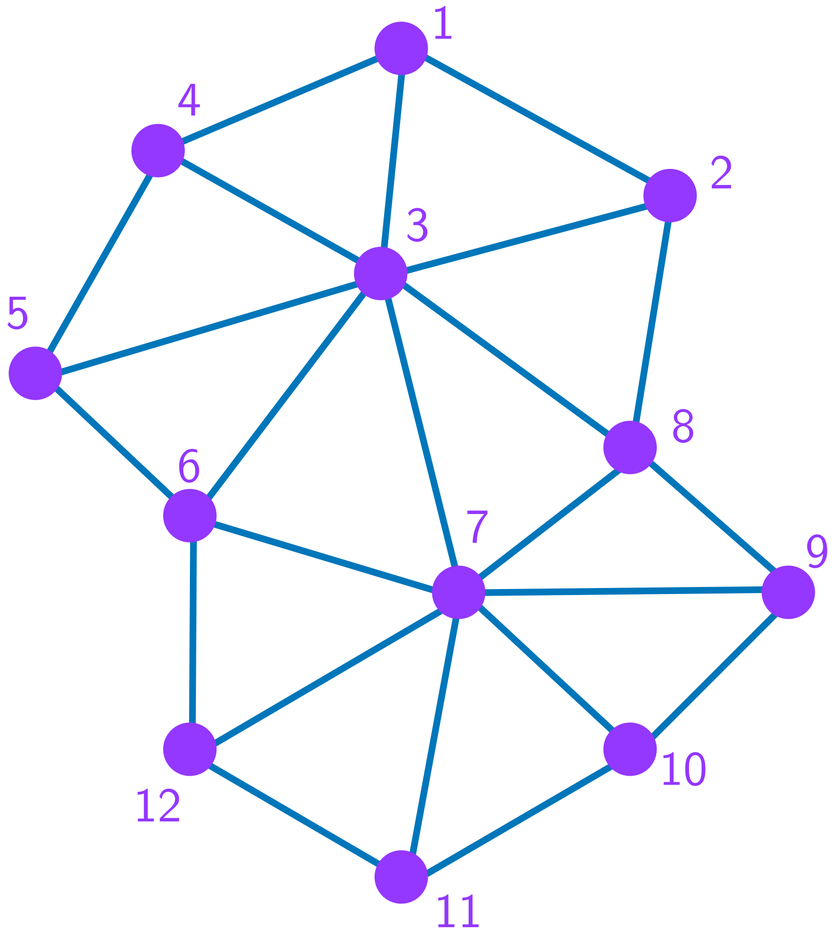
Sheet simulation: evaluating forces

- Simulation is built around ODE solver, using 6n degrees of freedom for n nodes
- Key routine in the simulation evaluates the RHS of the ODE, considering these contributions:
  - Damping
  - Self-contact forces
  - Stretching and bending forces
  - External forces



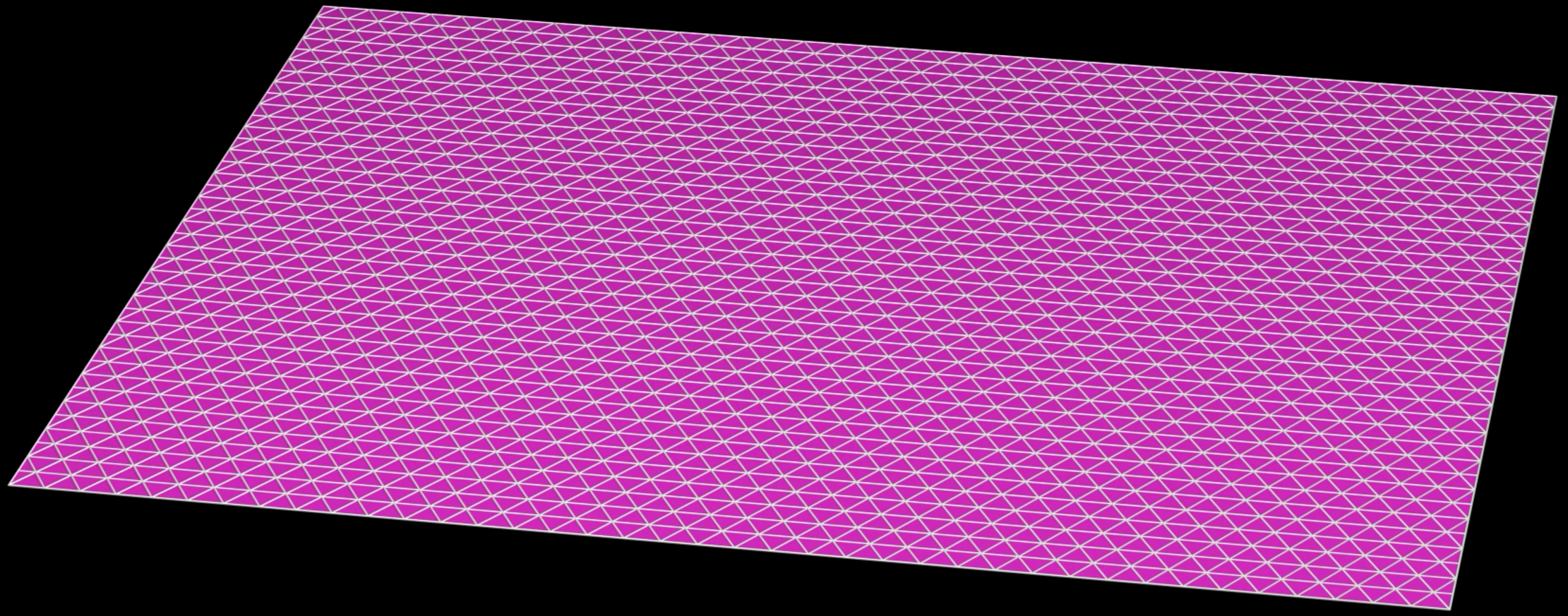
Sheet simulation: output

- Simulation outputs the positions and velocities of all nodes at fixed intervals (using dense output)
- A utility called **unpack** can convert these into a variety of text formats for plotting
- Can output the data into a format that can be read by POV-Ray, a freeware raytracer (www.povray.org)



Initial example

- 80x80 hexagonal mesh
- stretching only
- perturb spring rest lengths by $\pm$ 25%



N. Oppenheimer and T. A. Witten, Phys. Rev. E **92**, 052401 (2015).

Wrinkling

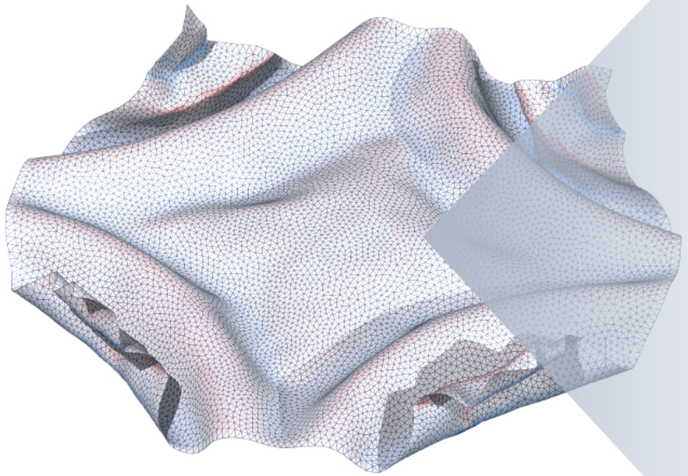
- Top-down view, colored by out-of-plane deformation
- Stretching and bending
- Spring nodes are loosely coupled to shrinking substrate

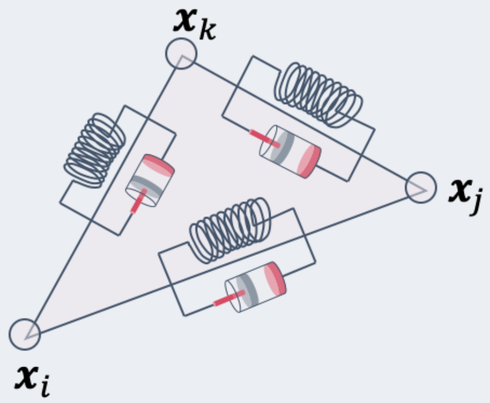
By Ann Bigelow
(UW–Madison Math)

Additional computational issues

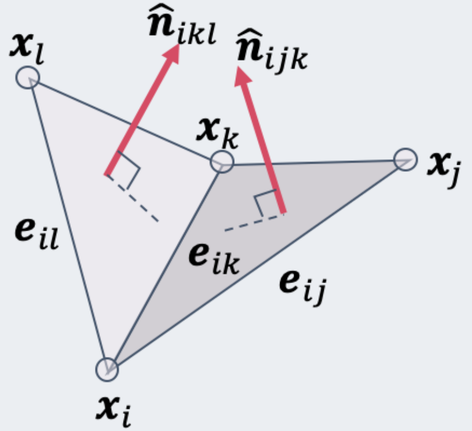
Two time-integration methods

Triangular mesh





in-plane interactions



out-of-plane rigidity

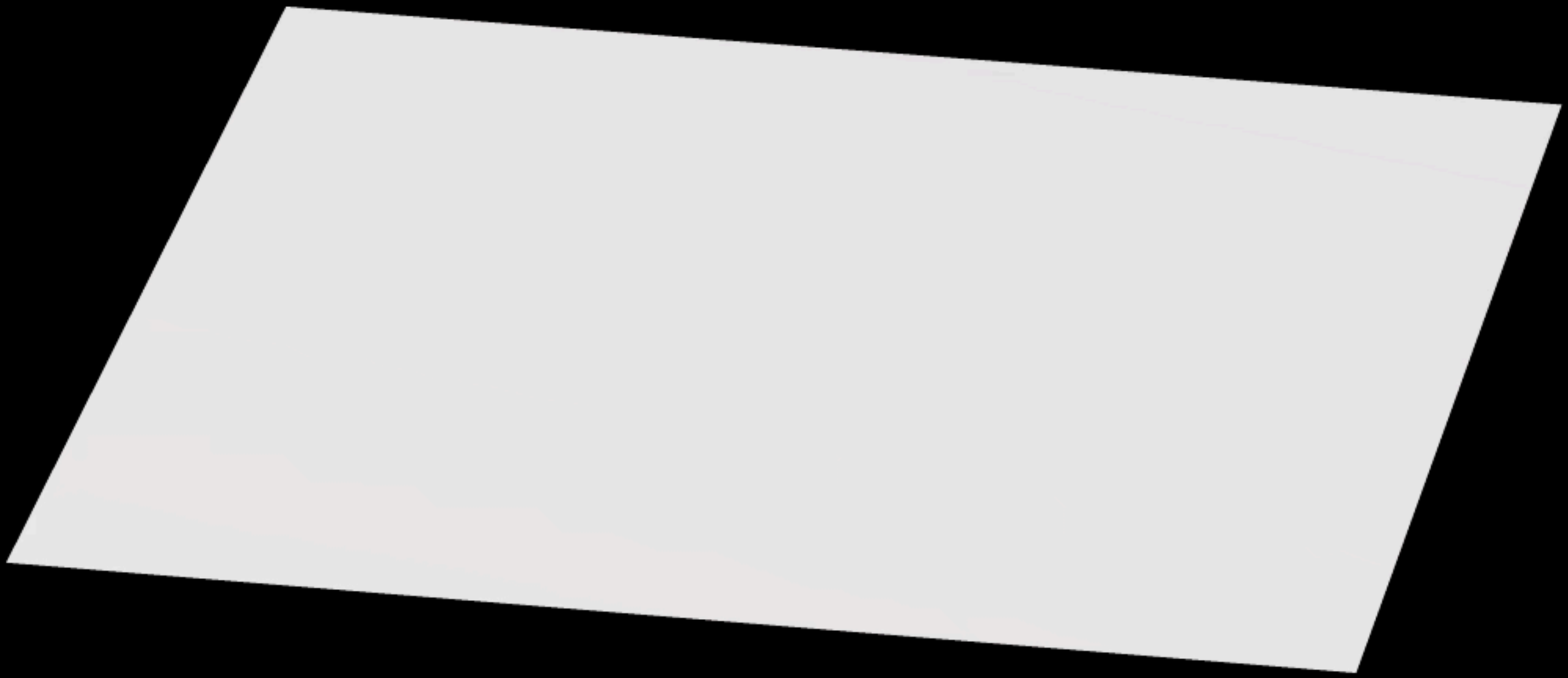
Explicit simulation

- Adaptive fourth-order Runge–Kutta method
- Uses FSAL (first same as last) method to construct third-order solution for step selection
- More efficient for regimes with many contact forces

Implicit simulation

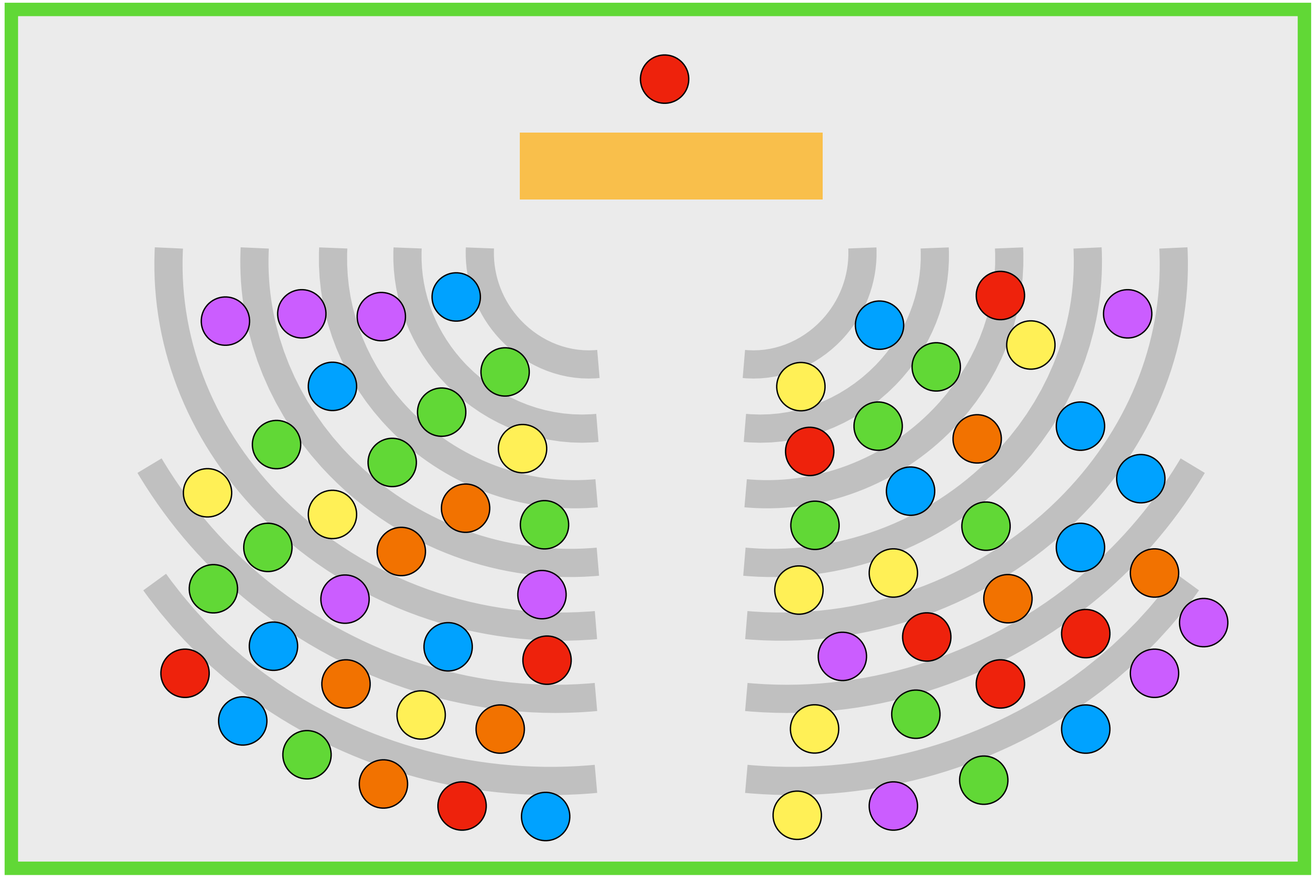
- Impose quasi-static constraint so that $\mathbf{F}_i = \mathbf{0}$
- Use third-order differential–algebraic solver
- Involves solving nonlinear system; use preconditioned conjugate gradient method for Newton steps

J. Andrejevic and C. H. Rycroft, *Simulation of crumpled sheets via alternating quasistatic and dynamic representations*, J. Comput. Phys. **471**, 111607 (2022).



Neighbor relations test

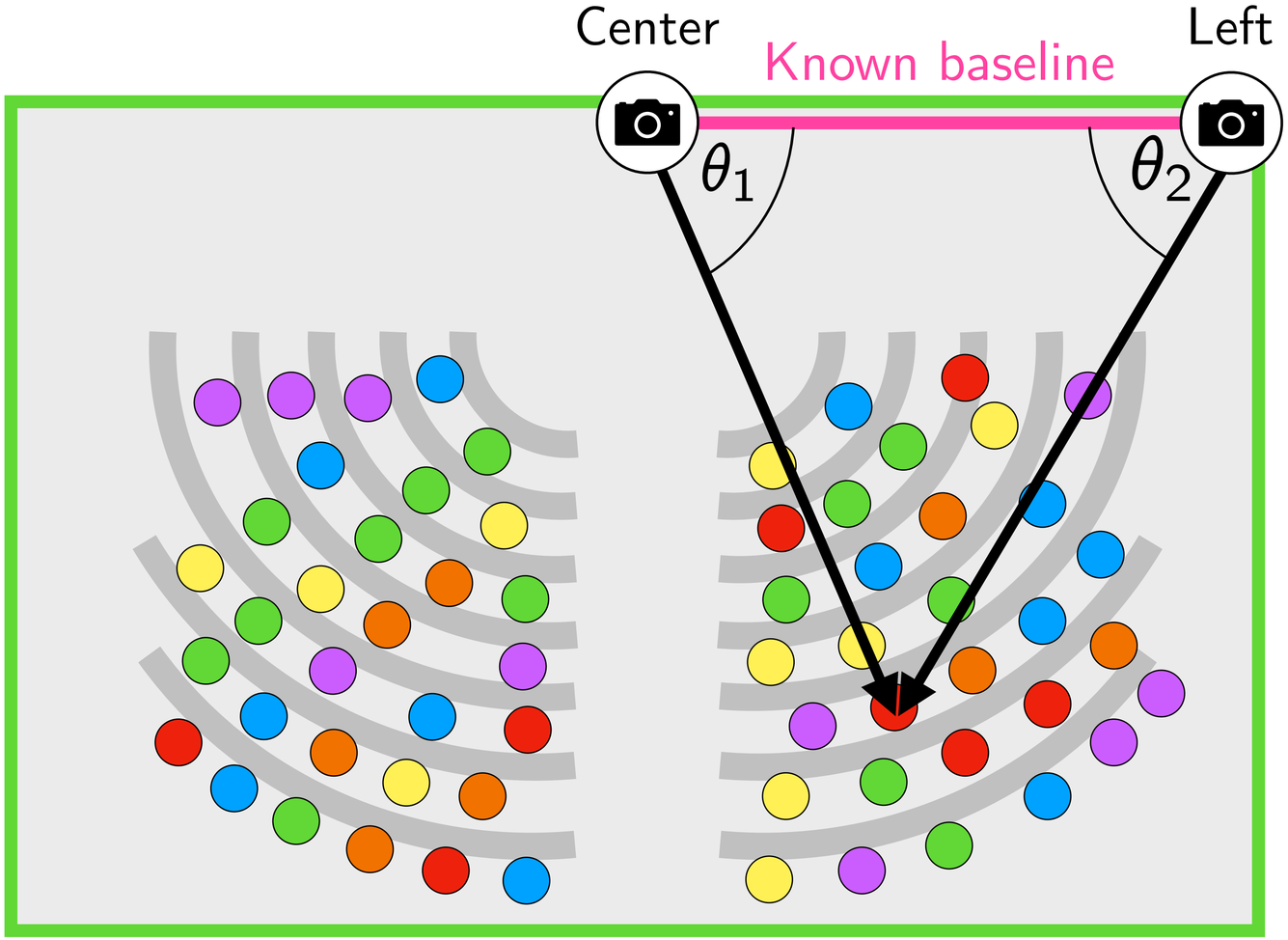
- In Tuesday's lecture we did an exercise to determine neighbor relationships using shared Voronoi edges
- I took several pictures from the front of the room, which is enough to determine everyone's position using triangulation



The lecture room

Triangulation

- Triangulation is split into two halves:
  - For the left side, use images on the left and in the center



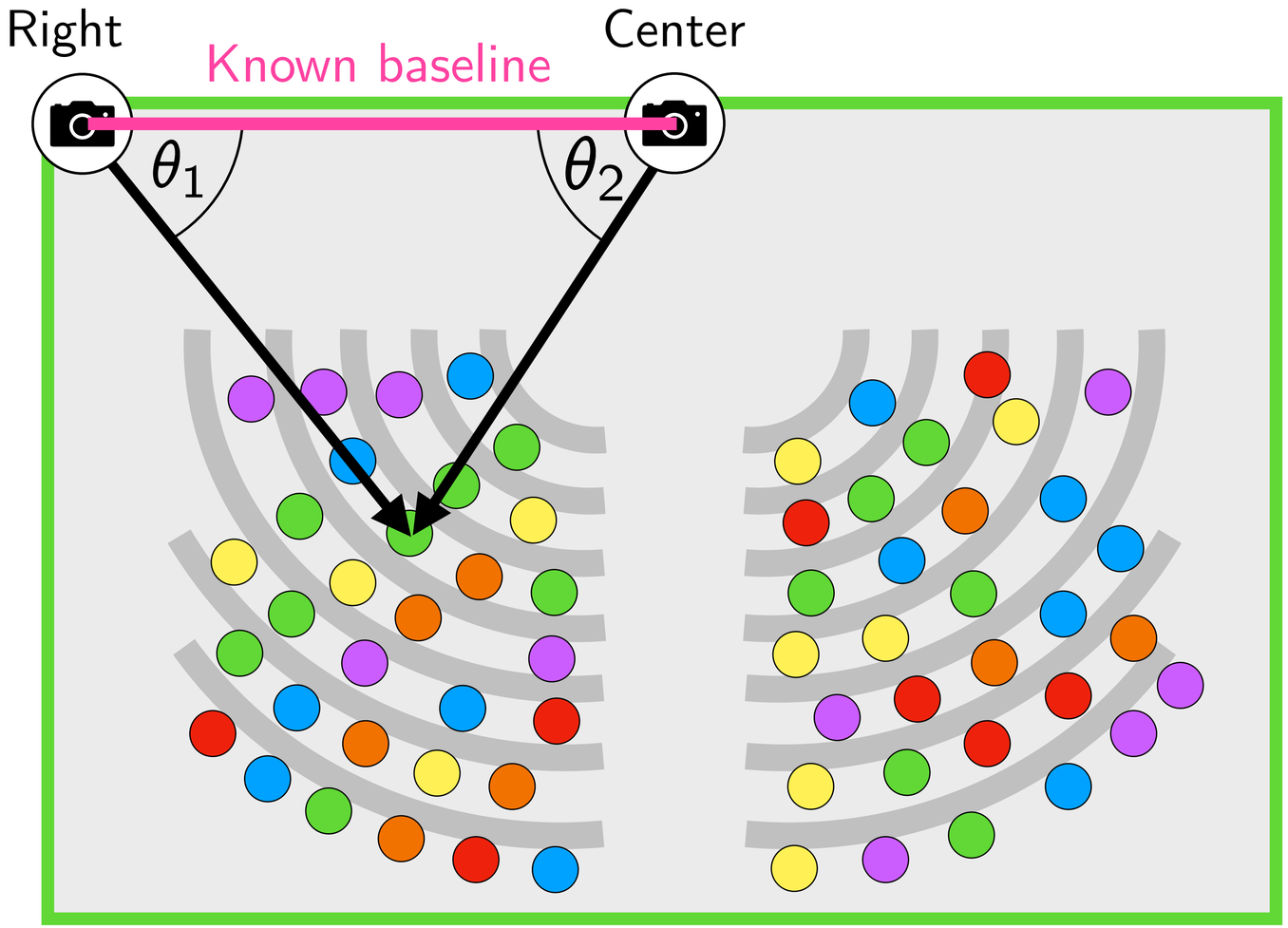
Right side

Left side

(From presenter's perspective)

Triangulation

- Triangulation is split into two halves:
  - For the left side, use images on the left and in the center
  - For the right side, use images in the center and on the right



(From presenter's perspective)


Left seating from left side

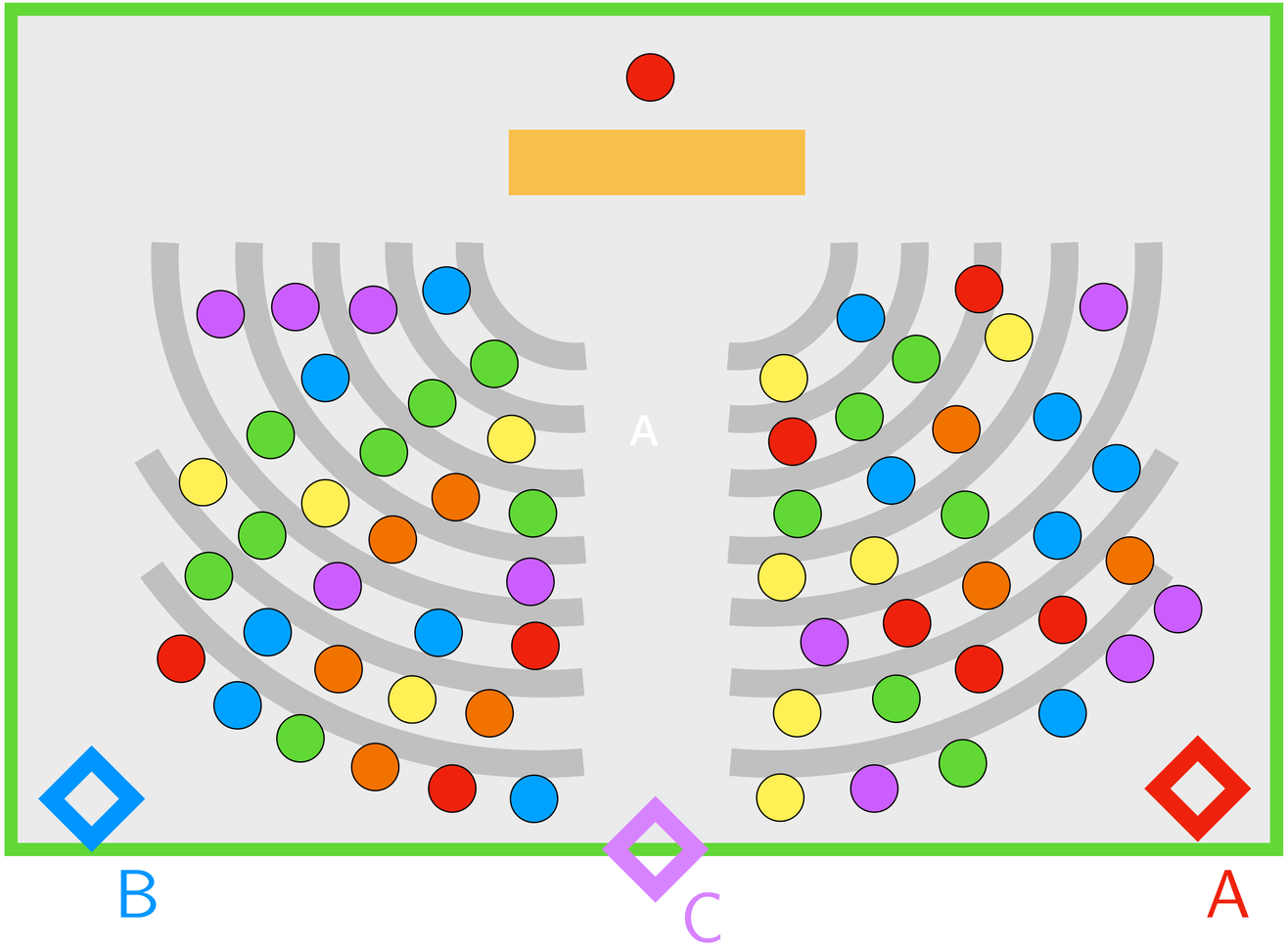

Left seating from center


Right seating from center


Right seating from right side

Determining camera orientation

- Three target points are used to find the precise angle and field of view
- They are A and B (room lights) and C (clock)
- At least two target points are visible in each photo, which is sufficient



Data collection

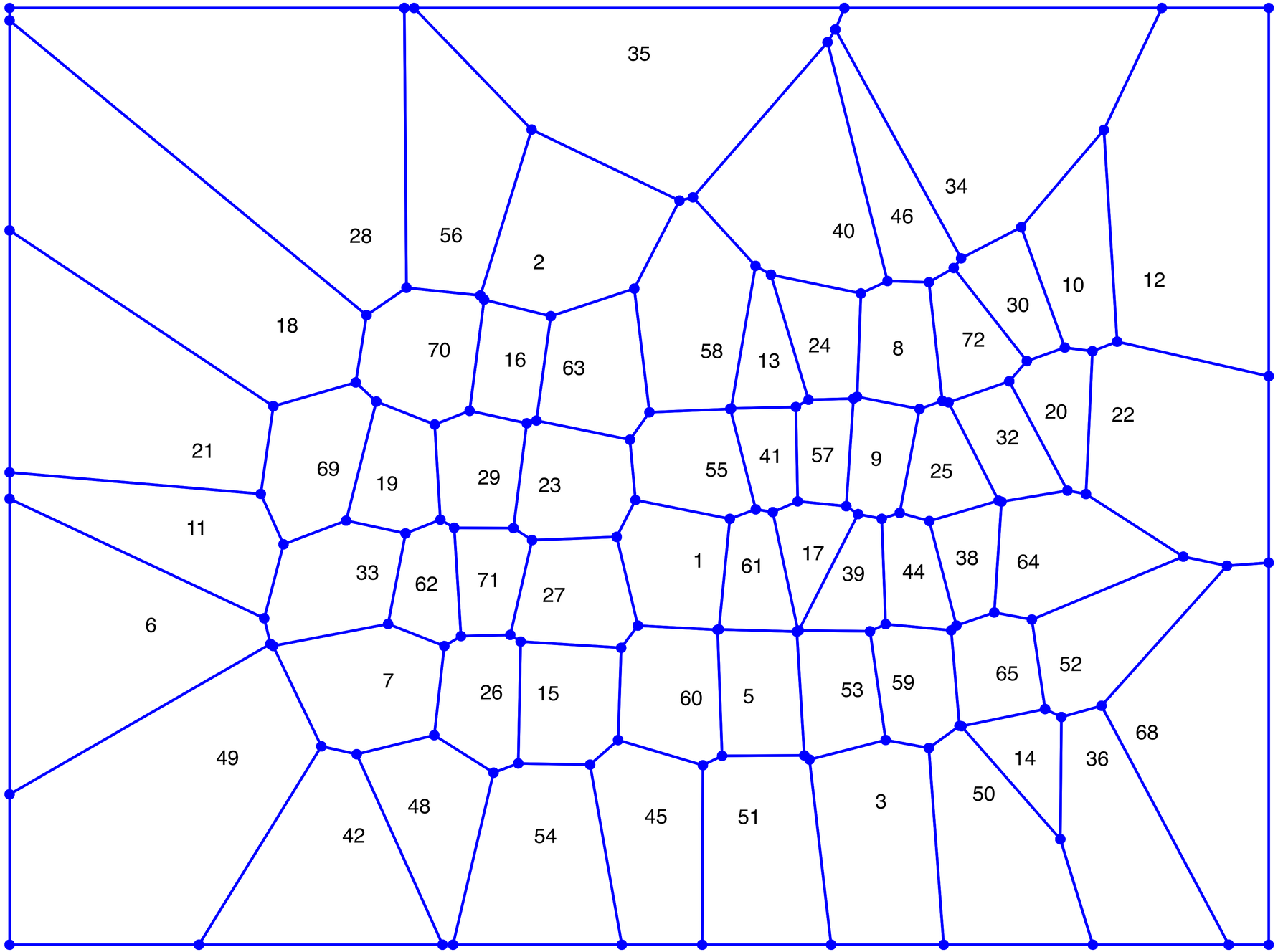
- Program files for performing the triangulation are in the **triangulate** directory of the Git repository
- The file **ang.dat** contains the horizontal pixel positions of each circle in each image
- The file has four sections labeled with headers IM0, IM1, IM2, IM3 for each image
- The header line contains the pixel positions of the three calibration targets

```
IM0 1007 3969 6093
68 1289
36 1706
52 2147
14 2305
22 2395
50 2520
12 2664
65 2669
64 2864
3 3169
20 3287
59 3406
38 3421
32 3666
53 3709
44 3778
51 3847
10 4043
25 4078
...
```

Neighbor data

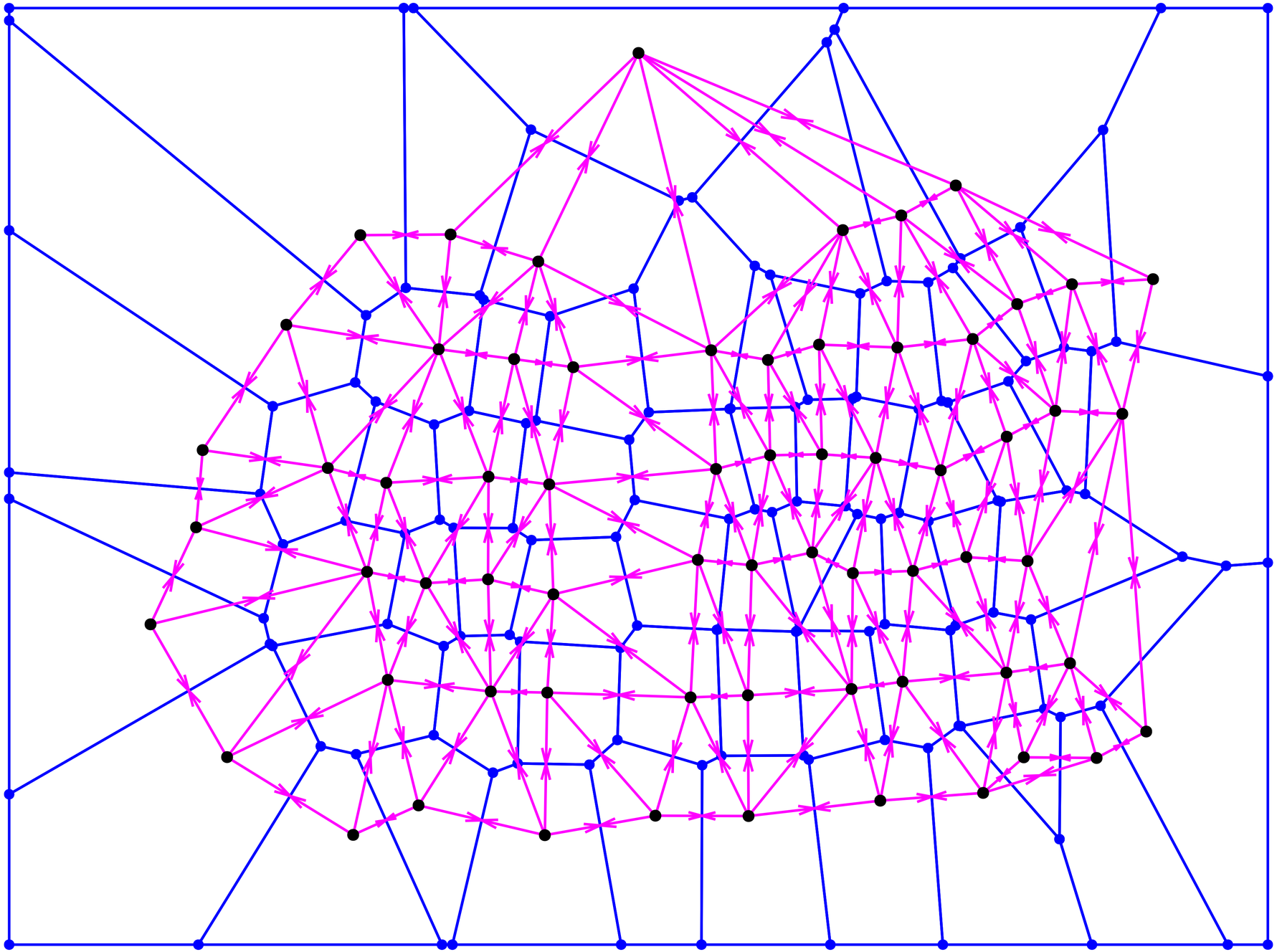
- Each participant submitted a note with
  - A list of neighbors based on visualizing their Voronoi cell and look at shared edges
  - An estimate of the Voronoi cell volume
- The data was entered into the file **nei.dat**
- See the triangulate directory in the GitHub repository for more information on analysis

Voronoi tessellation



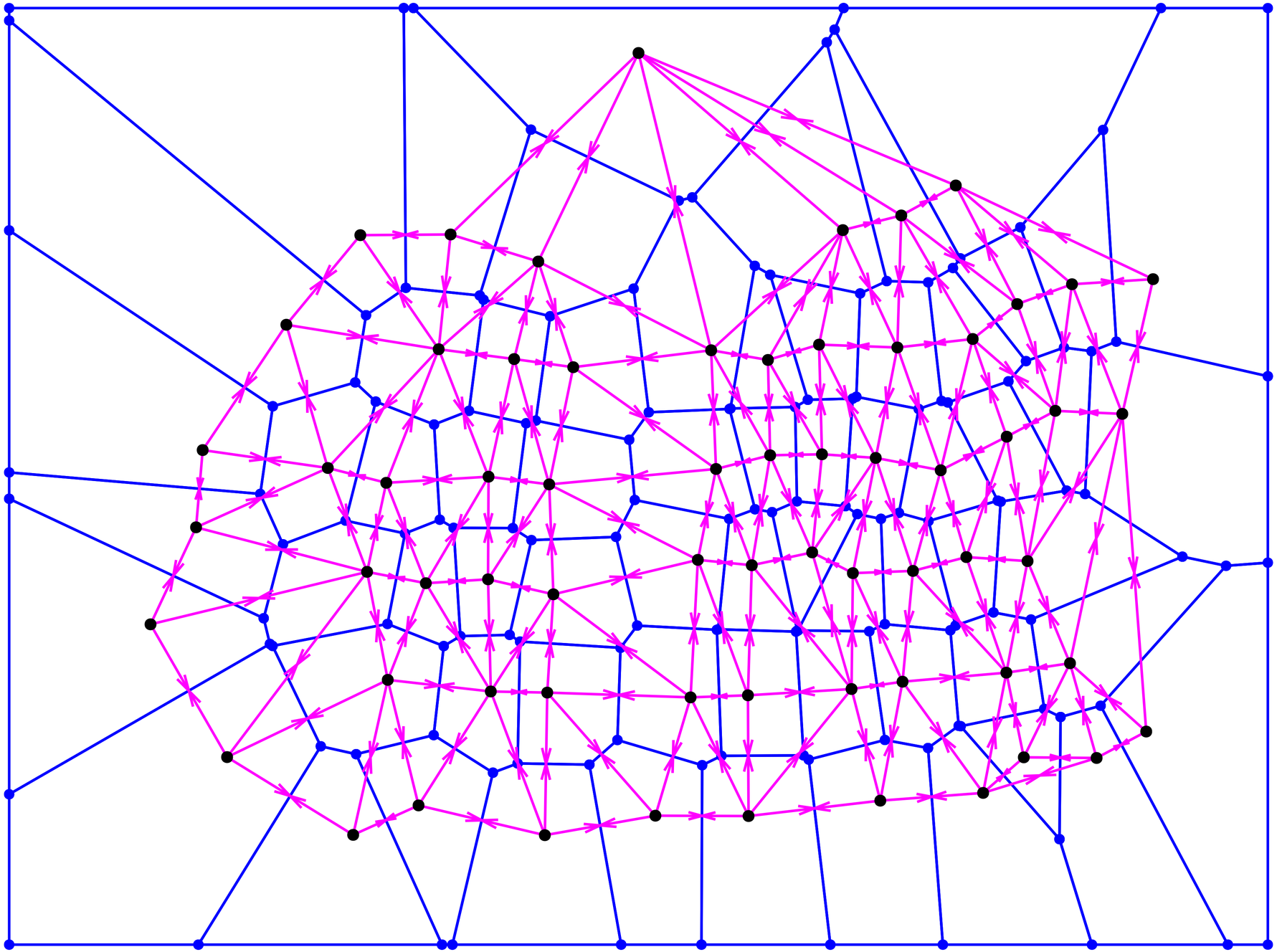
Neighbors

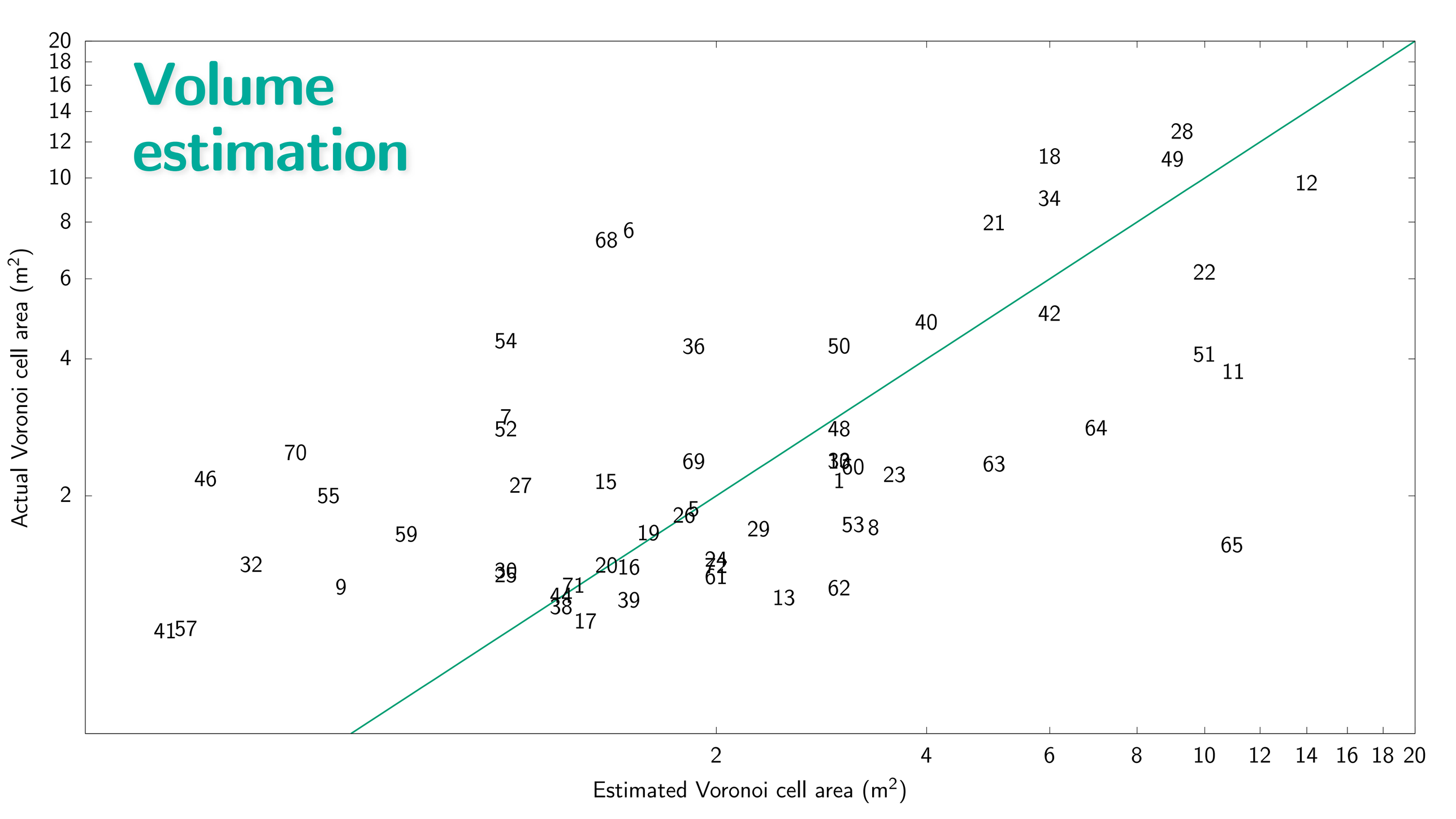
(from Delaunay triangulation)



Neighbors

(from submitted information)




Volume estimation
Actual Voronoi cell area (m²)
Estimated Voronoi cell area (m²)
20
18
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16
18
20
28
18
49
12
34
21
68
6
22
42
40
54
36
50
51
11
7
52
48
64
70
69
63
46
27
15
1
23
55
5
26
59
19
29
53
8
65
32
24
22
20
16
61
30
29
71
9
62
44
13
38
39
17
41
57

Smallest relative error magnitudes

ID number	Est. volume (m^2)	Voronoi volume (m^2)	Relative error
49	9	10.9510	-17.82%
40	4	4.7922	-16.53%
19	1.6	1.6461	-2.80%
71	1.25	1.2615	-0.91%
20	1.3935	1.3970	-0.25%
5	1.8581	1.8581	-0.054%
44	1.2	1.1994	0.047%
38	1.2	1.1323	5.98%
48	3	2.7905	7.51%
16	1.5	1.3839	8.39%

Supplementary info: differential-algebraic solver

DAE framework for thin sheet simulation

Newton's second law: $m\ddot{\mathbf{x}} = \mathbf{F}$ Order reduction: $\dot{\mathbf{x}} = \mathbf{v}$ (unit mass $m = 1$)

$\dot{\mathbf{v}} = \mathbf{F}$

Partial quasistatic regime: $\dot{\mathbf{x}} = \mathbf{v}$ *3N* differential equations

$\mathbf{F} = \mathbf{0}$ *3N* algebraic equations

Backward differentiation formula (BDF) of order p:

$$y(t_{n+1}) + \sum_{s=0}^{p-1} \alpha_{n-s} y(t_{n-s}) = \beta h y'(t_{n+1}, y(t_{n+1}))$$

Discretizing our differential equation:

$$-\mathbf{x}(t_{n+1}) - \sum_{s=0}^{p-1} \alpha_{n-s} \mathbf{x}(t_{n-s}) + \beta h \mathbf{v}(t_{n+1}) = 0$$

$$\mathbf{F}(t_{n+1}) = 0$$

DAE framework for thin sheet simulation

$$-\mathbf{x}(t_{n+1}) - \sum_{s=0}^{p-1} \alpha_{n-s}\mathbf{x}(t_{n-s}) + \beta h \mathbf{v}(t_{n+1}) = f^{\text{diff}}$$

$$\mathbf{F}(t_{n+1}) = f^{\text{alg}}$$

AM 205 Unit 4!

In general, these functions are nonlinear, and can be solved iteratively using Newton's method:

$$y_{k+1} = y_k - J_k^{-1}(y_k) f_k(y_k)$$
$$- J_k^{-1}(y_k)\Delta y_k = f_k(y_k)$$

Block matrix equation:

$$\begin{bmatrix} I & -\beta h I \\ \mathbf{H}^{\mathbf{x}} & \mathbf{H}^{\mathbf{v}} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_k \\ \Delta \mathbf{v}_k \end{bmatrix} = \begin{bmatrix} f_k^{\text{diff}} \\ f_k^{\text{alg}} \end{bmatrix}, \quad \begin{aligned} \mathbf{H}^{\mathbf{x}} &= -\nabla_{\mathbf{x}_k}\mathbf{F} \\ \mathbf{H}^{\mathbf{v}} &= -\nabla_{\mathbf{v}_k}\mathbf{F} \end{aligned}$$

$$(\mathbf{H}^{\mathbf{x}} + \frac{1}{\beta h}\mathbf{H}^{\mathbf{v}})\Delta \mathbf{x}_k = f_k^{\text{alg}} + \frac{1}{\beta h}\mathbf{H}^{\mathbf{v}} f_k^{\text{diff}}$$

$$\Delta \mathbf{v}_k = -\frac{1}{\beta h}(f_k^{\text{diff}} - \Delta \mathbf{x}_k)$$

$\mathbf{H}^{\mathbf{x}}$ is positive semi-definite in equilibrium, but not guaranteed at all time.

$$\mathbf{H}^{\mathbf{v}} \sim \frac{b_{\text{int}}}{\beta h k_s}\mathbf{H}^{\mathbf{x}} + \frac{b_{\text{iso}}}{\beta h} I$$

Isotropic damping term provides numerical stability through the strength of damping and the step size.

DAE framework for thin sheet simulation

$$\Delta \mathbf{v}_k = -\frac{1}{\beta h}(f_k^{\text{diff}} - \Delta \mathbf{x}_k)$$

can be solved by an algebraic step once $\Delta \mathbf{x}_k$ is known

$$(\mathbf{H}^{\mathbf{x}} + \frac{1}{\beta h}\mathbf{H}^{\mathbf{v}})\Delta \mathbf{x}_k = f_k^{\text{alg}} + \frac{1}{\beta h}\mathbf{H}^{\mathbf{v}} f_k^{\text{diff}}$$

is a large, sparse system $Ax = b$ with (hopefully) positive definite symmetric matrix A → solve iteratively via conjugate gradient (CG) method

AM 205 Unit 5!

Quick outline of algorithm:

- Adaptive step
- 3rd order BDF main solution, higher 4th order solution used for error estimation and step size selection

1. Ramp up orders to build solution history
2. While not yet reached final time:
3. Compute 4th order error estimator
4. Compute 3rd order candidate solution
5. Calculate error between solutions, accept or reject the step, and adjust step size

1. While not converged:
2. Newton iterations solve for $\mathbf{x}(t_{n+1}), \mathbf{v}(t_{n+1})$
3. While not converged:
4. CG iterations solve for $\Delta \mathbf{x}_k, \Delta \mathbf{v}_k$