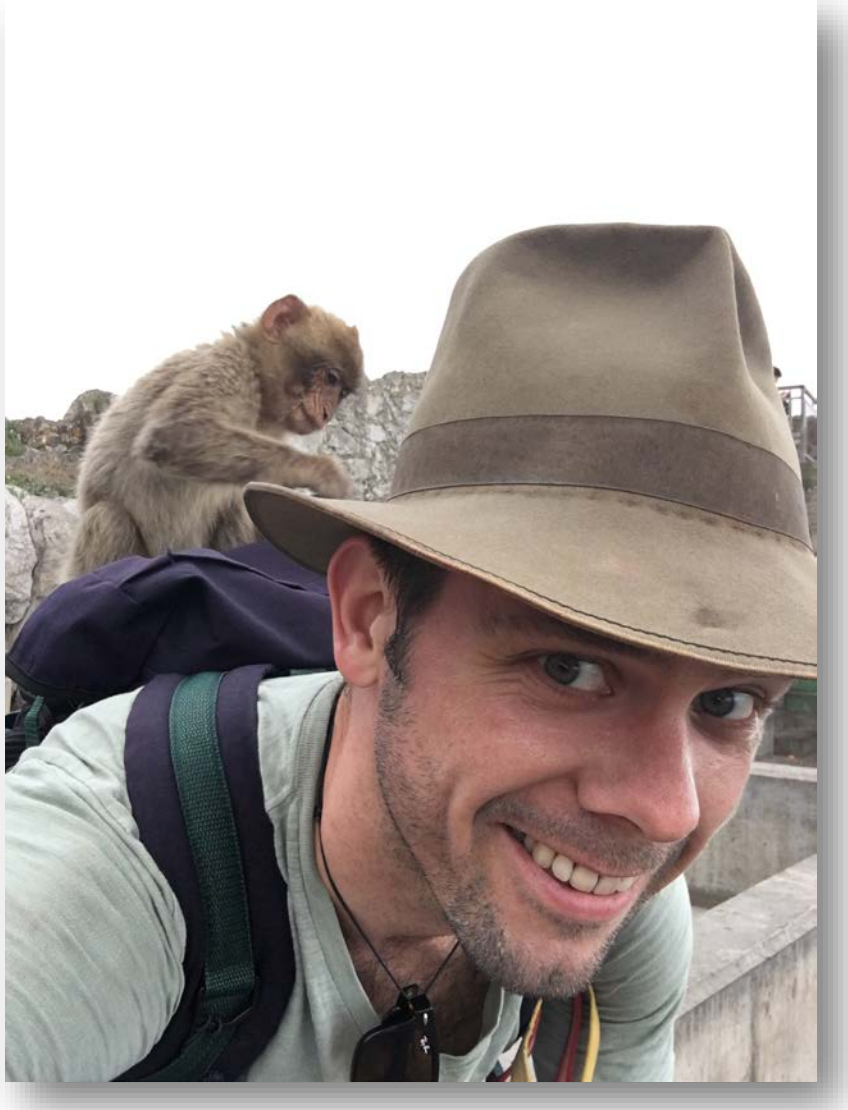
MATTE OG KONSERVERINGSLOVER

André R. Brodtkorb, Researcher
Department of Mathematics and Cybernetics
SINTEF Digital

Hvem er jeg?

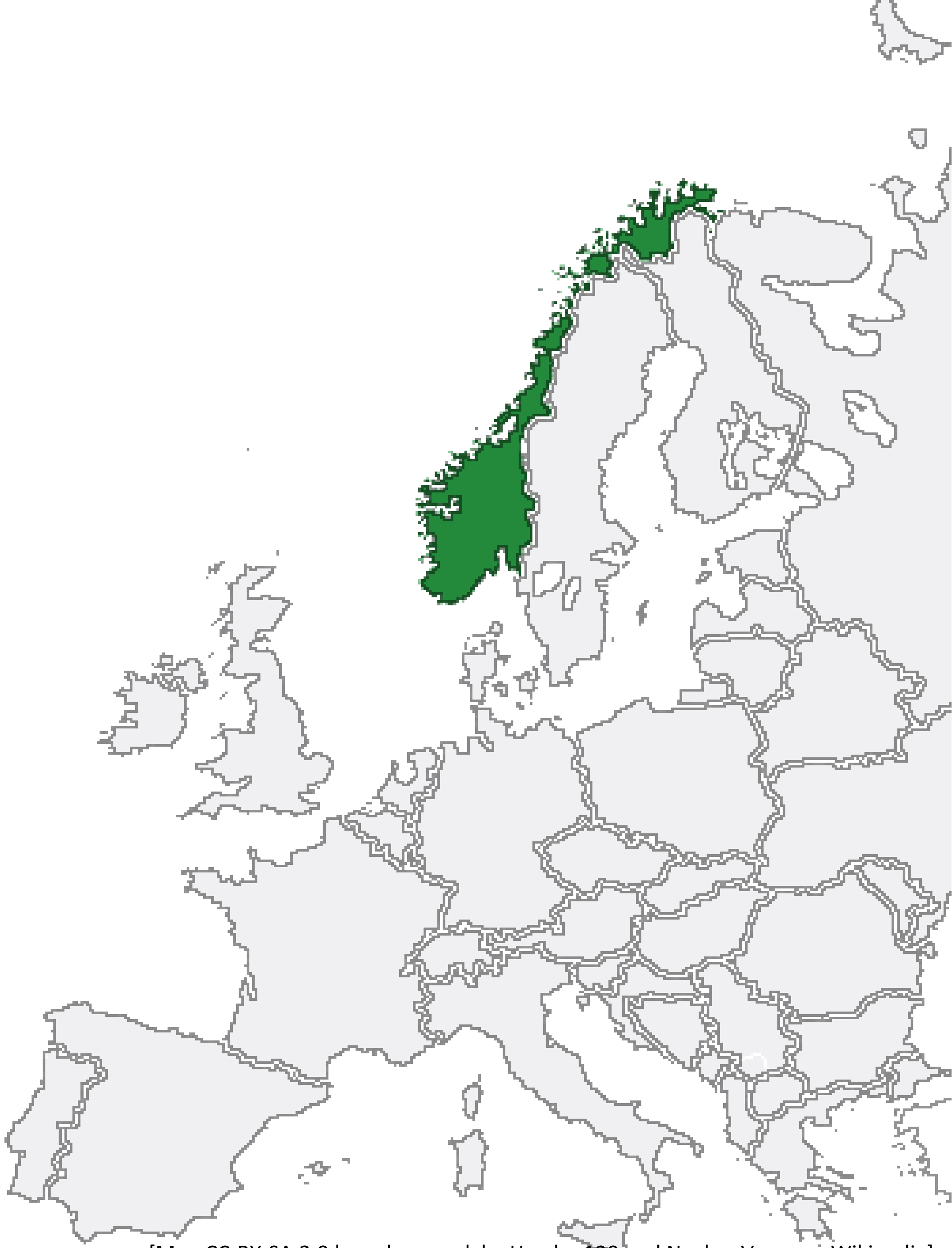


- Født og oppvokst i Drammen
- Driver med innebandy, tennis, løping, seiling, metallsløyd, … (ganske nerdete egentlig)
- Studert ved UiO, ferdig med doktorgrad i 2010
- Jobbet 10 år på SINTEF i Oslo (80% permisjon nå)
- Undervist på NITH (nå Westerdals), Universitetet i Oslo, og nå Høyskolen i Oslo

- Elsker koblingen mellom matte og data!

SINTEF

- Established 1950 by the Norwegian Institute of Technology.
- The largest independent research organisation in Scandinavia.
- A non-profit organisation.
- Motto: “Technology for a better society”.
- Key Figures*
  - 2100 Employees from 70 different countries.
  - 73% of employees are researchers.
  - 3 billion NOK in turnover (about 360 million EUR / 490 million USD).
  - 9000 projects for 3000 customers.
  - Offices in Norway, USA, Brazil, Chile, and Denmark.



[Map CC-BY-SA 3.0 based on work by Hayden120 and NuclearVacuum, Wikipedia]

Dagens (popvit) forelesning

- Litt om matte og data: Hvorfor trenger vi matte, og hvorfor trenger vi data?
- Litt arbeid jeg har jobbet med på SINTEF
- Litt om konserveringslover og programmering

Advarsel: En del videoer i dag ☺
(og litt "tung" matte)

History lesson: development of the microprocessor 1/2

1942: Digital Electric Computer

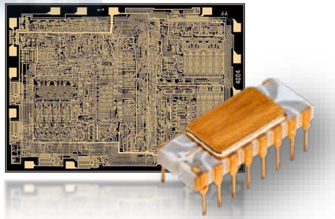
(Atanasoff and Berry)

1947: Transistor

(Shockley, Bardeen, and Brattain)

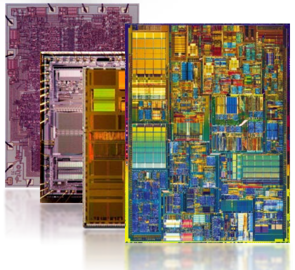
1958: Integrated Circuit

(Kilby)



1971: Microprocessor

(Hoff, Faggin, Mazor)



1971- Exponential growth

(Moore, 1965)

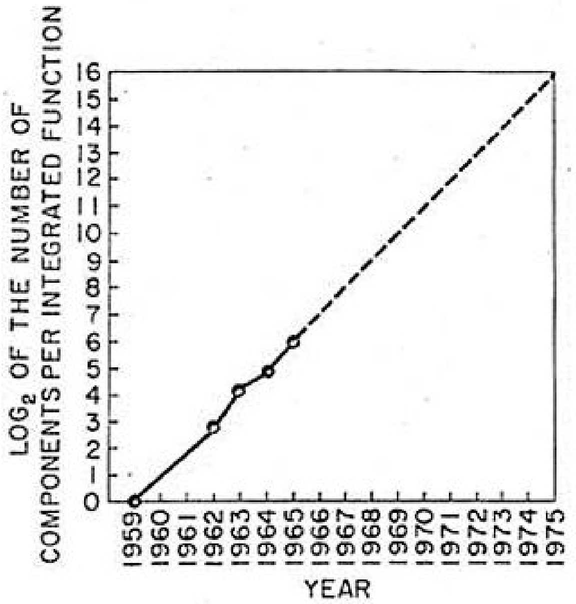
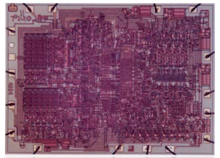
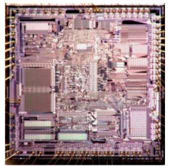


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

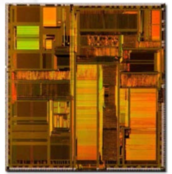
History lesson: development of the microprocessor 2/2



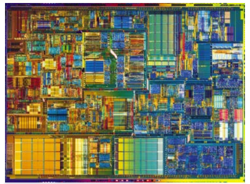
1971: 4004,
2300 trans, 740 KHz



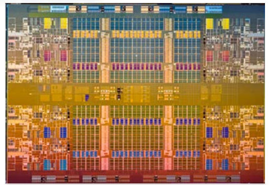
1982: 80286,
134 thousand trans, 8 MHz



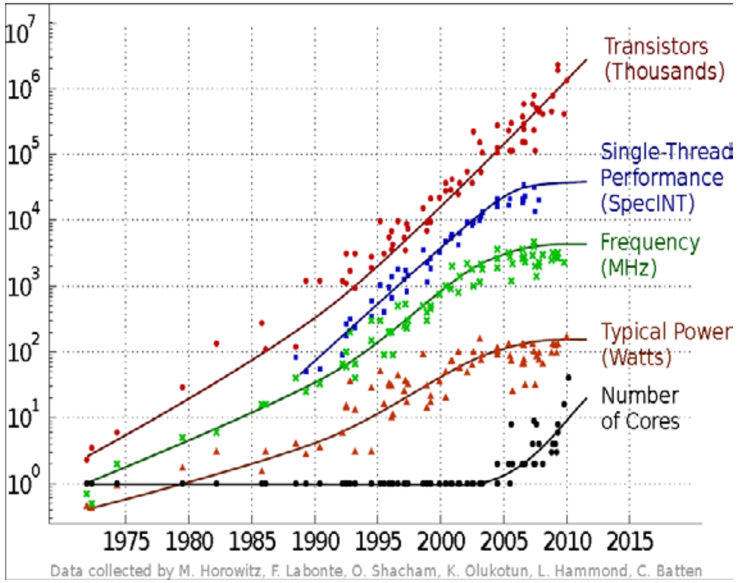
1993: Pentium P5,
1.18 mill. trans, 66 MHz



2000: Pentium 4,
42 mill. trans, 1.5 GHz



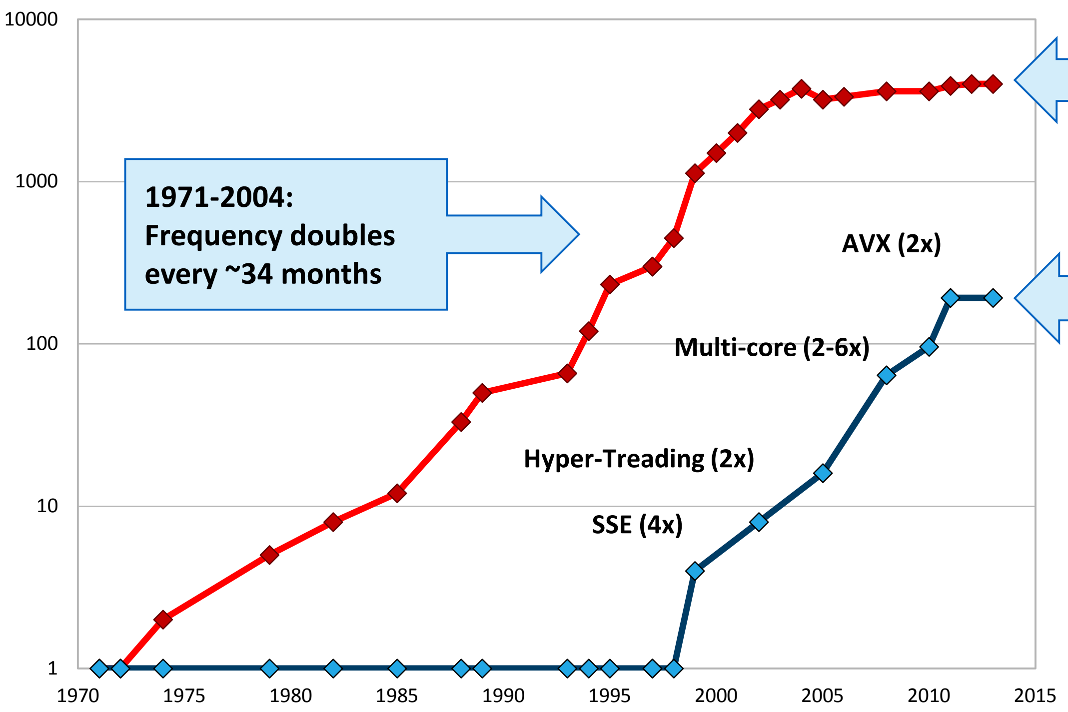
2010: Nehalem
2.3 bill. Trans, 8 cores, 2.66 GHz



Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten

End of frequency scaling

Desktop processor performance (SP)



1971-2004:
Frequency doubles
every ~34 months

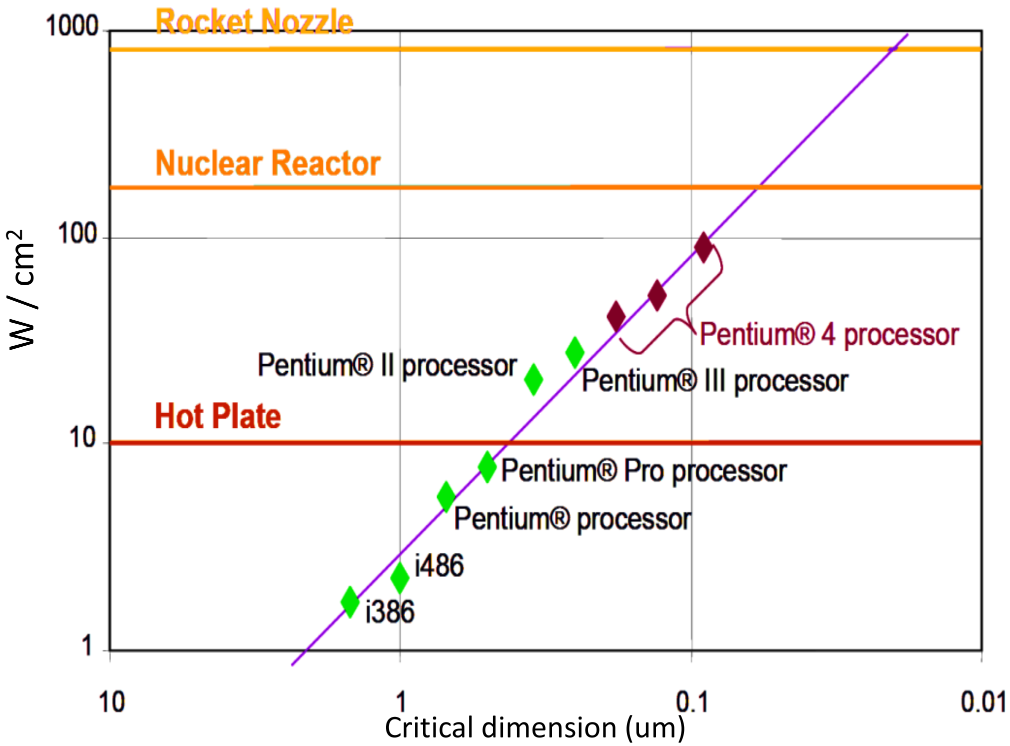
2004-2014:
Frequency
constant

1999-2014:
Parallelism doubles
every ~30 months

- 1970-2004: Frequency doubles every 34 months (Moore’s law for performance)
- 1999-2014: Parallelism doubles every 30 months

What happened in 2004?

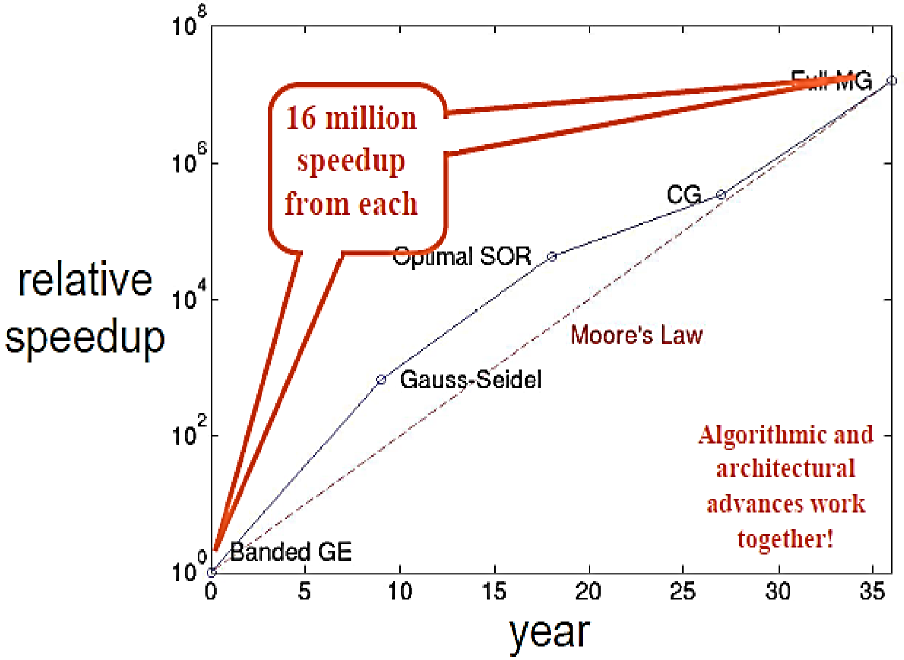
- Heat density approaching that of nuclear reactor core: Power wall
- Traditional cooling solutions (heat sink + fan) insufficient
- Industry solution: multi-core and parallelism!



Graph taken from G. Taylor, “Energy Efficient Circuit Design and the Future of Power Delivery” EPEPS’09

Why care about mathematics?

- The key to increasing performance, is to consider the full algorithm and architecture interaction.
- A good knowledge of both the algorithm and the computer architecture is required.



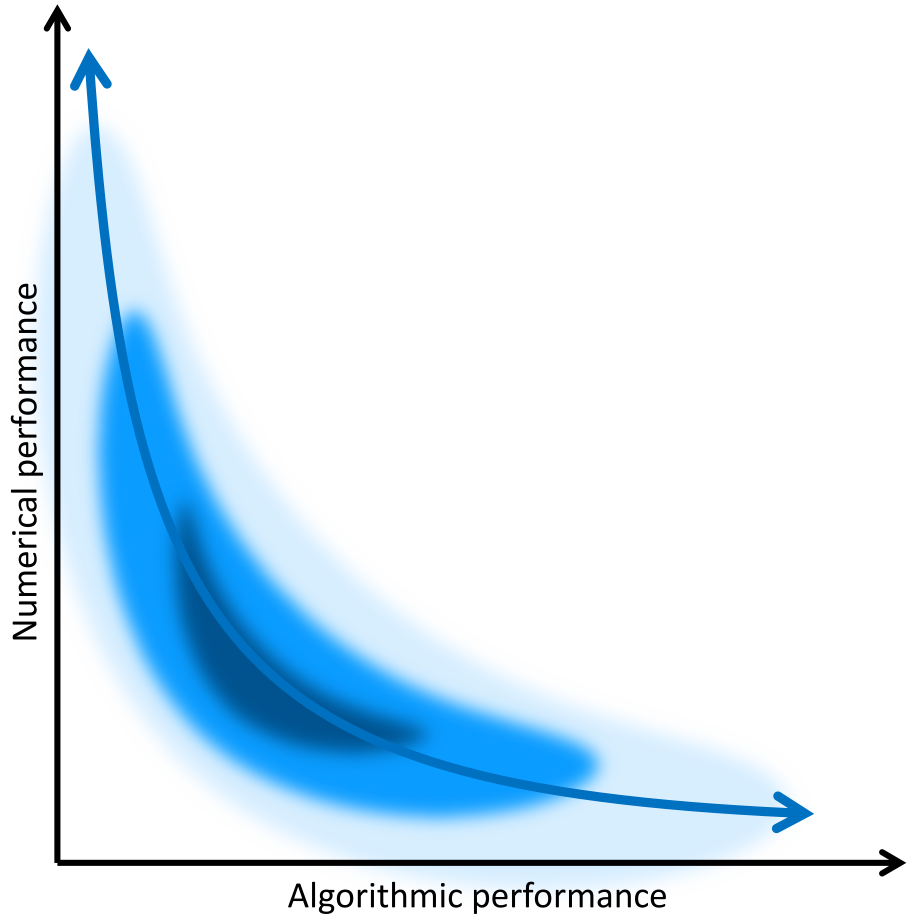
16 million speedup from each

Algorithmic and architectural advances work together!

Graph from David Keyes, Scientific Discovery through Advanced Computing, Geilo Winter School, 2008

Algorithmic and numerical performance

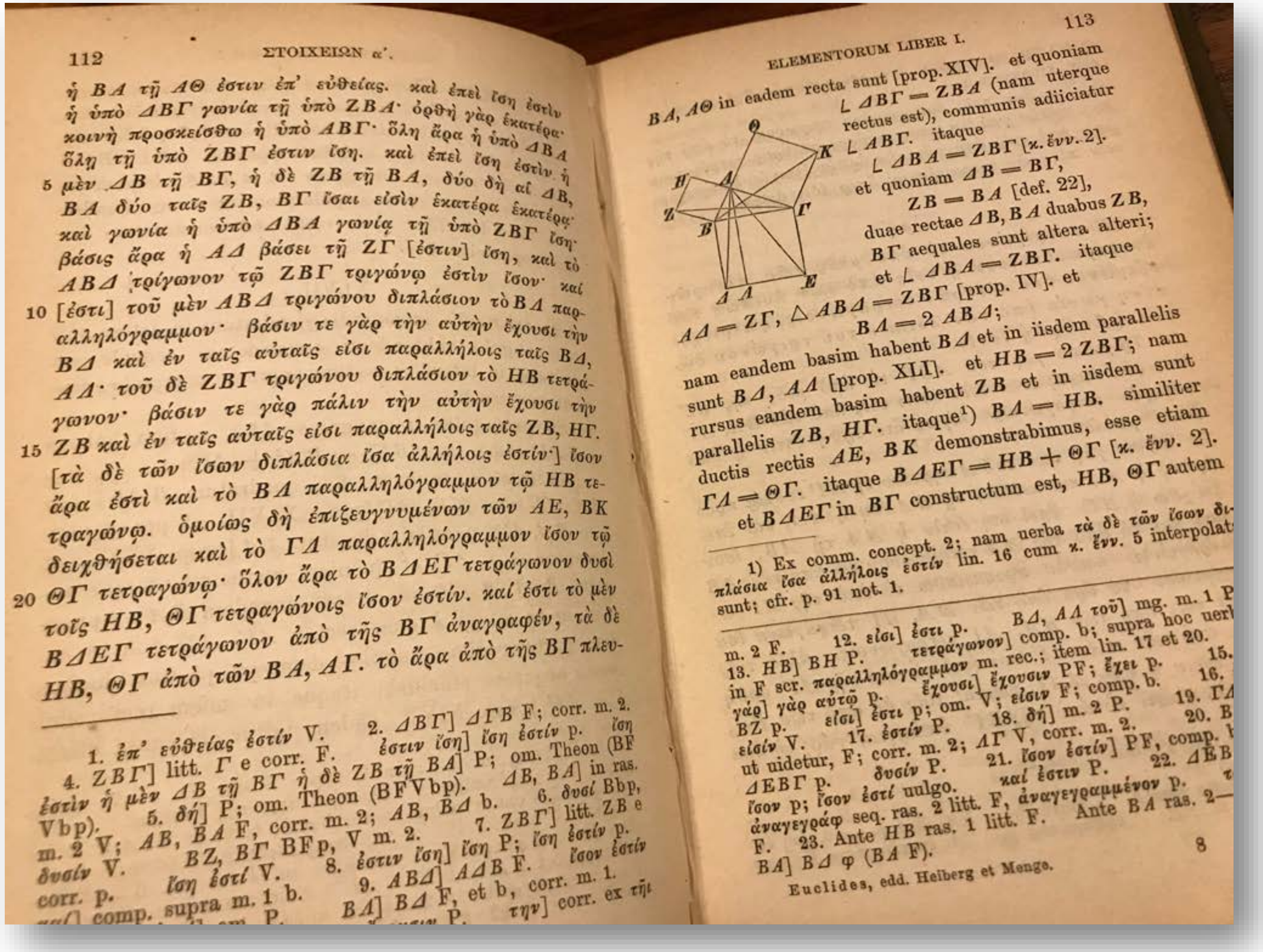
- Total performance is the product of algorithmic **and** numerical performance
- Your mileage may vary: algorithmic performance is highly problem dependent
- Many algorithms have low numerical performance
- Need to consider both the algorithm and the architecture for maximum performance



Når matte og data ikke spiller på lag

Matte er helt gresk for meg

- Euklids Elementa (300 fkr)



112 ΣΤΟΙΧΕΙΩΝ α΄.

ἡ ΒΑ τῇ ΑΘ ἐστιν ἐπ᾽ εὐθείας. καὶ ἐπεὶ ἴση ἐστὶν ἡ ὑπὸ ΔΒΓ γωνία τῇ ὑπὸ ΖΒΑ· ὀρθὴ γὰρ ἑκατέρα· κοινὴ προσκείσθω ἡ ὑπὸ ΑΒΓ· ὅλη ἄρα ἡ ὑπὸ ΔΒΑ ὅλῃ τῇ ὑπὸ ΖΒΓ ἐστιν ἴση. καὶ ἐπεὶ ἴση ἐστὶν ἡ μὲν ΔΒ τῇ ΒΓ, ἡ δὲ ΖΒ τῇ ΒΑ, δύο δὴ αἱ ΔΒ, ΒΑ δύο ταῖς ΖΒ, ΒΓ ἴσαι εἰσὶν ἑκατέρα ἑκατέρᾳ· καὶ γωνία ἡ ὑπὸ ΔΒΑ γωνίᾳ τῇ ὑπὸ ΖΒΓ ἴση· βάσις ἄρα ἡ ΑΔ βάσει τῇ ΖΓ [ἐστιν] ἴση, καὶ τὸ ΑΒΔ τρίγωνον τῷ ΖΒΓ τριγώνῳ ἐστὶν ἴσον· καὶ [ἔστι] τοῦ μὲν ΑΒΔ τριγώνου διπλάσιον τὸ ΒΛ παραλληλόγραμμον· βάσιν τε γὰρ τὴν αὐτὴν ἔχουσι τὴν ΒΔ καὶ ἐν ταῖς αὐταῖς εἰσι παραλλήλοις ταῖς ΒΔ, ΑΛ· τοῦ δὲ ΖΒΓ τριγώνου διπλάσιον τὸ ΗΒ τετράγωνον· βάσιν τε γὰρ πάλιν τὴν αὐτὴν ἔχουσι τὴν ΖΒ καὶ ἐν ταῖς αὐταῖς εἰσι παραλλήλοις ταῖς ΖΒ, ΗΓ. [τὰ δὲ τῶν ἴσων διπλάσια ἴσα ἀλλήλοις ἐστίν·] ἴσον ἄρα ἐστὶ καὶ τὸ ΒΛ παραλληλόγραμμον τῷ ΗΒ τετραγώνῳ. ὁμοίως δὴ ἐπιζευγνυμένων τῶν ΑΕ, ΒΚ δειχθήσεται καὶ τὸ ΓΛ παραλληλόγραμμον ἴσον τῷ ΘΓ τετραγώνῳ· ὅλον ἄρα τὸ ΒΔΕΓ τετράγωνον δυσὶ τοῖς ΗΒ, ΘΓ τετραγώνοις ἴσον ἐστίν. καί ἐστι τὸ μὲν ΒΔΕΓ τετράγωνον ἀπὸ τῆς ΒΓ ἀναγραφέν, τὰ δὲ ΗΒ, ΘΓ ἀπὸ τῶν ΒΑ, ΑΓ. τὸ ἄρα ἀπὸ τῆς ΒΓ πλευ-

---

1. ἐπ᾽ εὐθείας ἐστίν V. 2. ΔΒΓ] ΔΓΒ F; corr. m. 2. 4. ΖΒΓ] litt. Γ e corr. F. ἐστιν ἴση] ἴση ἐστίν p. ἴση ἐστὶν ἡ μὲν ΔΒ τῇ ΒΓ ἡ δὲ ΖΒ τῇ ΒΑ] P; om. Theon (BFVbp). 5. δή] P; om. Theon (BFVbp). ΔΒ, ΒΑ] in ras. m. 2 V; ΑΒ, ΒΑ F, corr. m. 2; ΑΒ, ΒΔ b. 6. δυσί Bbp, δυσίν V. ΒΖ, ΒΓ BFp, V m. 2. 7. ΖΒΓ] litt. ΖΒ e corr. p. ἴση ἐστί V. 8. ἐστιν ἴση] ἴση P; ἴση ἐστίν p. ἴσον ἐστίν comp. supra m. 1 b. 9. ΑΒΔ] ΑΔΒ F. ΒΛ] ΒΔ F, et b, corr. m. 1. τὴν] corr. ex τῇ

ELEMENTORUM LIBER I. 113

$B\Lambda$, $A\Theta$ in eadem recta sunt [prop. XIV]. et quoniam $\angle \Delta B\Gamma = ZBA$ (nam uterque rectus est), communis adiiciatur $\angle AB\Gamma$. itaque $\angle \Delta BA = ZB\Gamma$ [κ. ἔνν. 2]. et quoniam $\Delta B = B\Gamma$, $ZB = BA$ [def. 22], duae rectae ΔB, BA duabus ZB, $B\Gamma$ aequales sunt altera alteri; et $\angle \Delta BA = ZB\Gamma$. itaque $A\Delta = Z\Gamma$, $\triangle AB\Delta = ZB\Gamma$ [prop. IV]. et $B\Lambda = 2\,AB\Delta$; nam eandem basim habent $B\Delta$ et in iisdem parallelis sunt $B\Delta$, $A\Lambda$ [prop. XLI]. et $HB = 2\,ZB\Gamma$; nam rursus eandem basim habent ZB et in iisdem sunt parallelis ZB, $H\Gamma$. itaque[1] $B\Lambda = HB$. similiter ductis rectis AE, BK demonstrabimus, esse etiam $\Gamma\Lambda = \Theta\Gamma$. itaque $B\Delta E\Gamma = HB + \Theta\Gamma$ [κ. ἔνν. 2]. et $B\Delta E\Gamma$ in $B\Gamma$ constructum est, HB, $\Theta\Gamma$ autem

---

[1] Ex comm. concept. 2; nam uerba τὰ δὲ τῶν ἴσων διπλάσια ἴσα ἀλλήλοις ἐστίν lin. 16 cum κ. ἔνν. 5 interpolat sunt; cfr. p. 91 not. 1.

---

m. 2 F. 12. εἰσι] ἐστι p. ΒΔ, ΑΛ τοῦ] mg. m. 1 P
13. ΗΒ] ΒΗ P. τετράγωνον] comp. b; supra hoc uer
in F scr. παραλληλόγραμμον m. rec.; item lin. 17 et 20. 15.
γάρ] γὰρ αὐτῷ p. ἔχουσι] ἔχουσιν PF; ἔχει p. 16.
ΒΖ p. εἰσι] ἐστι p; om. V; εἰσιν F; comp. b. 19. ΓΛ
εἰσίν V. 17. ἐστίν P. 18. δή] m. 2 P. 20. Β
ut uidetur, F; corr. m. 2; ΛΓ V, corr. m. 2. 21. ἴσον ἐστίν] PF, comp.
ΔΕΒΓ p. δυσίν P. καί ἐστιν P. 22. ΔΕΒ
ἴσον p; ἴσον ἐστί uulgo. ἀναγεγραμμένον p.
ἀναγεγράφ seq. ras. 2 litt. F, Ante ΒΑ ras. 2—
F. 23. Ante ΗΒ ras. 1 litt. F.
ΒΑ] ΒΔ φ (ΒΑ F).

Euclides, edd. Heiberg et Menge. 8

The patriot missile…

- Designed by Raytheon (US) as an air defense system.
- Designed for time-limited use (up-to 8 hours) in mobile locations.
- Heavily used as static defenses using the Gulf war.
- Failed to intercept an incoming Iraqi Scud missile in 1991.
- 28 killed, 98 injured.

The patriot missile…

- It appears, that 0.1 seconds is not really 0.1 seconds…
  - Especially if you add a large amount of them

```
Python:

> print 0.1
0.1
> print "%.10f" % 0.1
0.1000000000
> print "%.20f" % 0.1
0.10000000000000000555
> print "%.30f" % 0.1
0.100000000000000005551115123126
```

Hours	Inaccuracy (sec)	Approx. shift in Range Gate (meters)
0	0	0
1	.0034	7
8	.0025	55
20	.0687	137
48	.1648	330
72	.2472	494
100	.3433	687

http://sydney.edu.au/engineering/it/~alum/patriot_bug.html

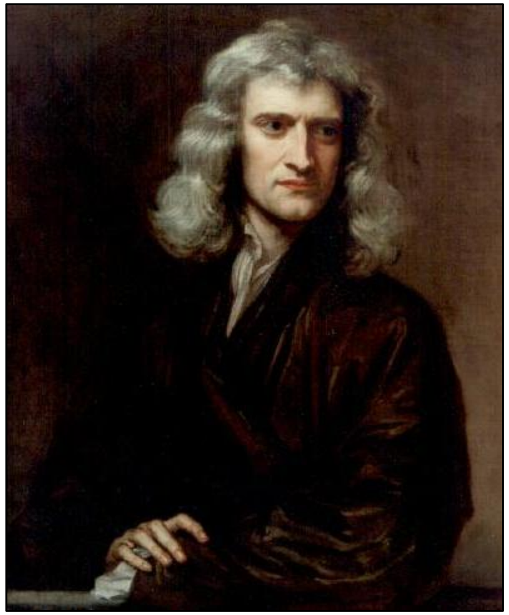
Konserveringslover

Konserveringslover - bevaringslover

- Konservere – bevare

- Eksempel: Mengden vann vil ikke endres, men være konstant

Conservation Laws


Isaac Newton, by Gottfried Kneller, public domain

- A conservation law describes that a quantity is conserved
- Comes from the physical laws of nature
- Example: Newtons first law: When viewed in an inertial reference frame, an object either remains at rest or continues to move at a constant velocity, unless acted upon by an external force.
- Example: Newtons third law: When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.
- More examples: conservation of mass (amount of water) in shallow water, amount of energy (heat) in the heat equation, linear momentum, angular momentum, etc.
- Conservation laws are mathematically formulated as partial differential equations: PDEs

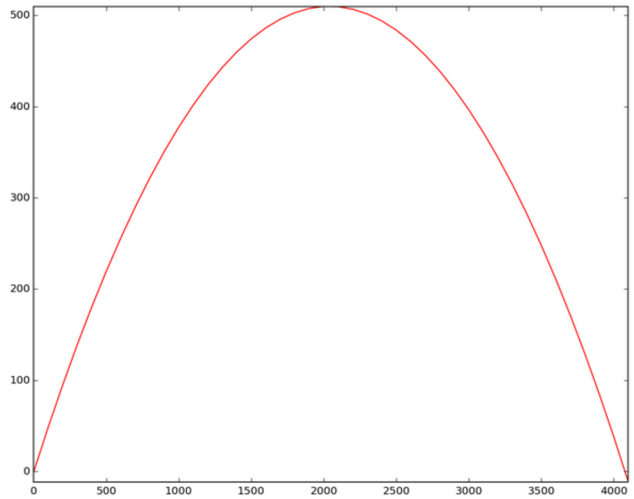
Ordinary Differential Equations (ODEs)

- Let us look at Newtons second law
  - The vector sum of the external forces F on an object is equal to the mass m of that object multiplied by the acceleration vector a of the object:
  - $\vec{F} = m \cdot \vec{a}$

- We know that acceleration, a, is the rate of change of speed over time, or in other words
  - $a = v' = \frac{dv}{dt}$

- We can then write Newtons second law as an ODE:
  - $F = m\frac{dv}{dt}$

Trajectory of a projectile



- From Newton's second law, we can derive a simple ODE for the trajectory of a projectile

  - Acceleration due to gravity:
    - $\vec{a} = [0, 0, 9.81]$
  - Velocity as a function of time
    - $\vec{v}(t) = \overrightarrow{v_o} + t \cdot \vec{a}$
  - Change in position, p, over time is a function of the velocity
    - $\frac{d\vec{p}}{dt} = \vec{v}(t)$

  - We can solve this ODE analytically with pen and paper, but for more complex ODEs, that becomes infeasible

  - The term "computer" used to be the profession for those who (amongst other things) calculated advanced projectile trajectories (air friction etc.).

Solving a simple ODE numerically

- To solve the ODE numerically on a computer, we discretize it

- To discretize an ODE is to replace the continuous derivatives with discrete derivatives, and to impose a discrete grid.

- In our ODE, we discretize in time, so that

$$\frac{d\vec{p}}{dt} = \vec{v}(t)$$

becomes

$$\frac{\vec{p}^{n+1} - \vec{p}^{n}}{\Delta t} = \vec{v}(n \cdot \Delta t)$$

Here, Δt is the grid spacing in time, and superscript n denotes the time step

Initial conditions

- Recall our discretization

$$\frac{\vec{p}^{n+1} - \vec{p}^{n}}{\Delta t} = \vec{v}(n \cdot \Delta t)$$

Rewriting so that n+1 is on the left hand side, we get an explicit formula

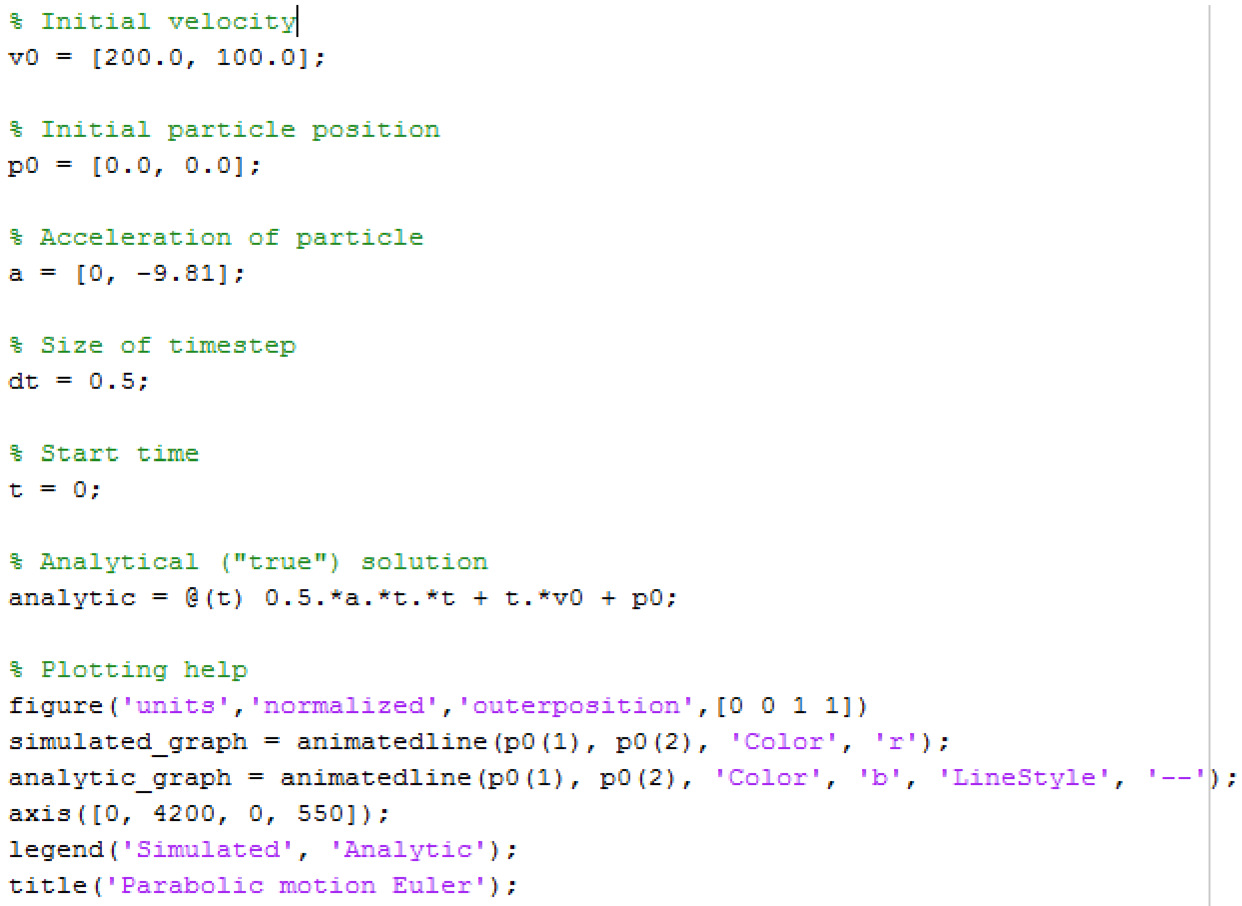
$$\vec{p}^{n+1} = \vec{p}^{n} + \Delta t \cdot \vec{v}(n \cdot \Delta t)$$

- Given initial conditions, that is the initial position, p^0, and the initial velocity, v^0, we can now simulate!

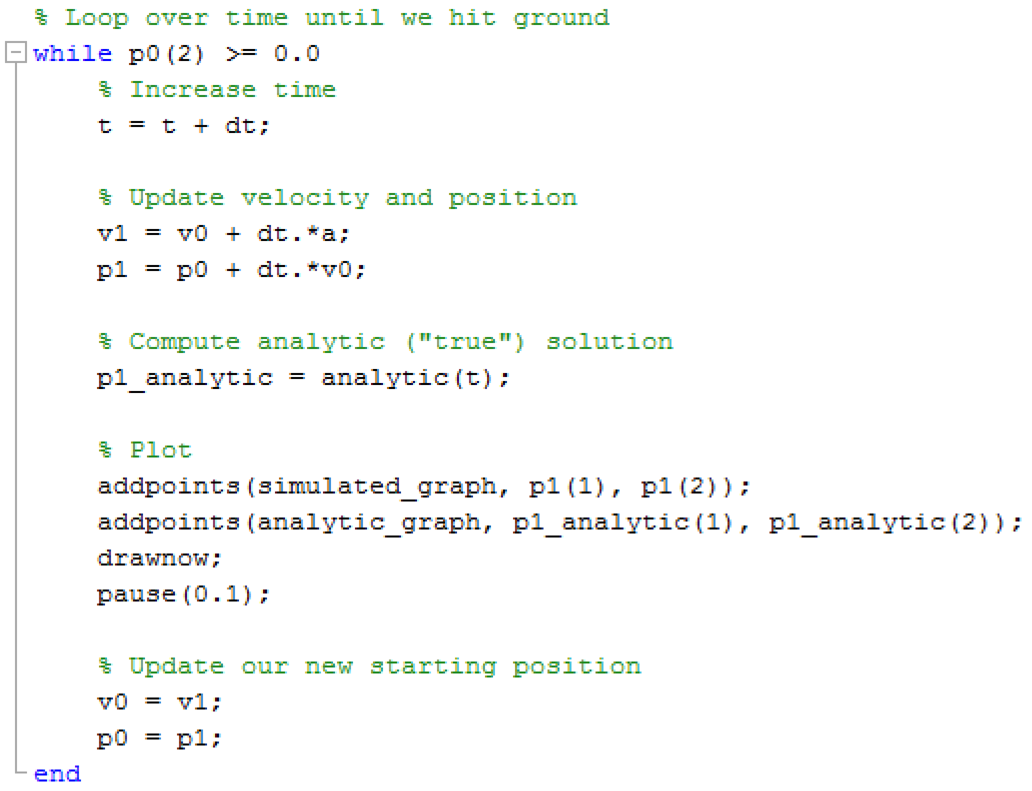
  - Example:

t	p	v
0	0.0	0.0
0.1	p0 + dt*v0 = 0.0	v0 - t*9.81 = -0.981
0.2	p1 - dt*v1 = -0.0981	v0 - t*9.81 = -1.962
0.2	…	…

Particle projectory in Matlab



```matlab
% Initial velocity
v0 = [200.0, 100.0];

% Initial particle position
p0 = [0.0, 0.0];

% Acceleration of particle
a = [0, -9.81];

% Size of timestep
dt = 0.5;

% Start time
t = 0;

% Analytical ("true") solution
analytic = @(t) 0.5.*a.*t.*t + t.*v0 + p0;

% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = animatedline(p0(1), p0(2), 'Color', 'r');
analytic_graph = animatedline(p0(1), p0(2), 'Color', 'b', 'LineStyle', '--');
axis([0, 4200, 0, 550]);
legend('Simulated', 'Analytic');
title('Parabolic motion Euler');
```



```matlab
% Loop over time until we hit ground
while p0(2) >= 0.0
    % Increase time
    t = t + dt;

    % Update velocity and position
    v1 = v0 + dt.*a;
    p1 = p0 + dt.*v0;

    % Compute analytic ("true") solution
    p1_analytic = analytic(t);

    % Plot
    addpoints(simulated_graph, p1(1), p1(2));
    addpoints(analytic_graph, p1_analytic(1), p1_analytic(2));
    drawnow;
    pause(0.1);

    % Update our new starting position
    v0 = v1;
    p0 = p1;
end
```

Particle trajectory results

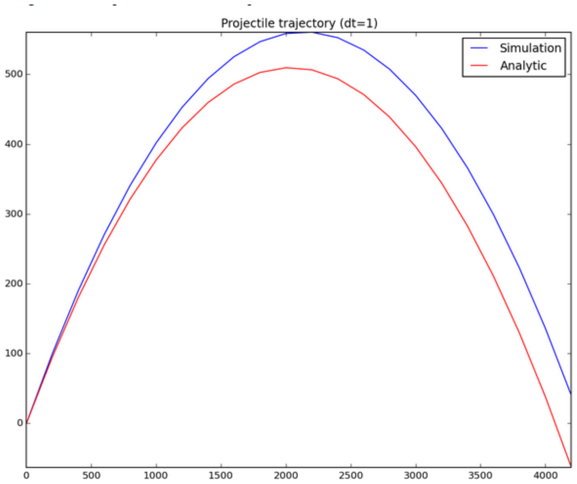
- When writing simulator code it is essential to check for correctness.

- The analytical solution to our problem is

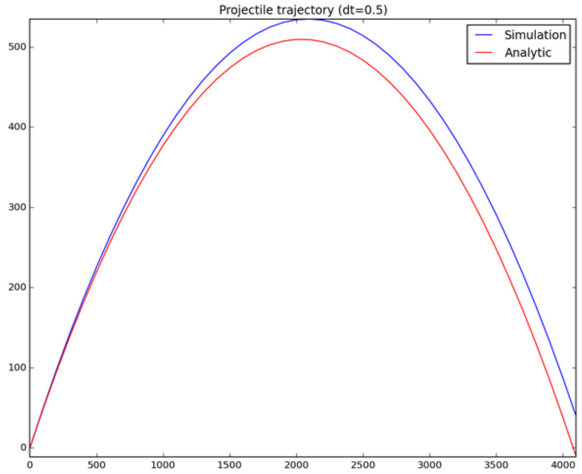
$$p(t) = \frac{1}{2}\vec{a}t^2 + t \cdot v^0 + p^0$$

- Let us compare the solutions

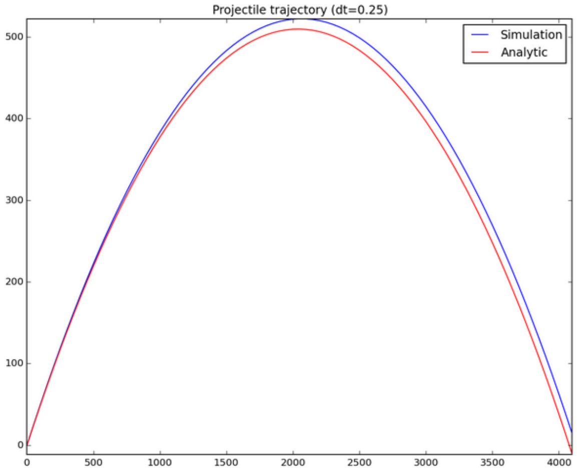
dt=1



dt=0.5



dt=0.25



More accuracy

- We have used a very simple integration rule (or approximation to the derivative)
  - Our rule is known as forward Euler

$$p^{n+1} = p^n + \Delta t \cdot \vec{v}$$

- We can get much higher accuracy with more advanced techniques such as Runge-Kutta 2

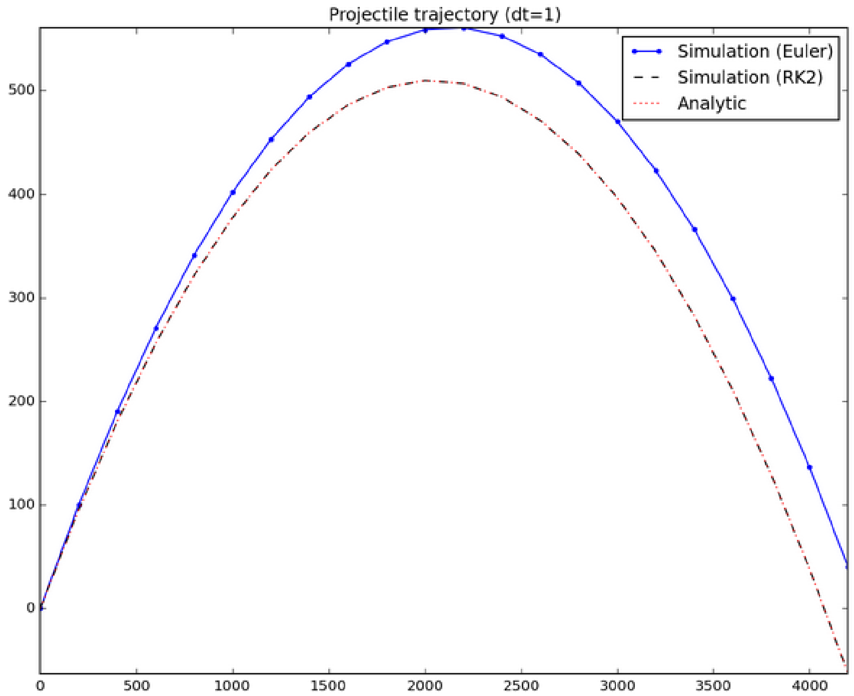
$$p^{*} = p^n + \Delta t \cdot \vec{v}(n \cdot \Delta t)$$

$$p^{**} = p^{*} + \Delta t \cdot \vec{v}\big((n+1) \cdot \Delta t\big)$$

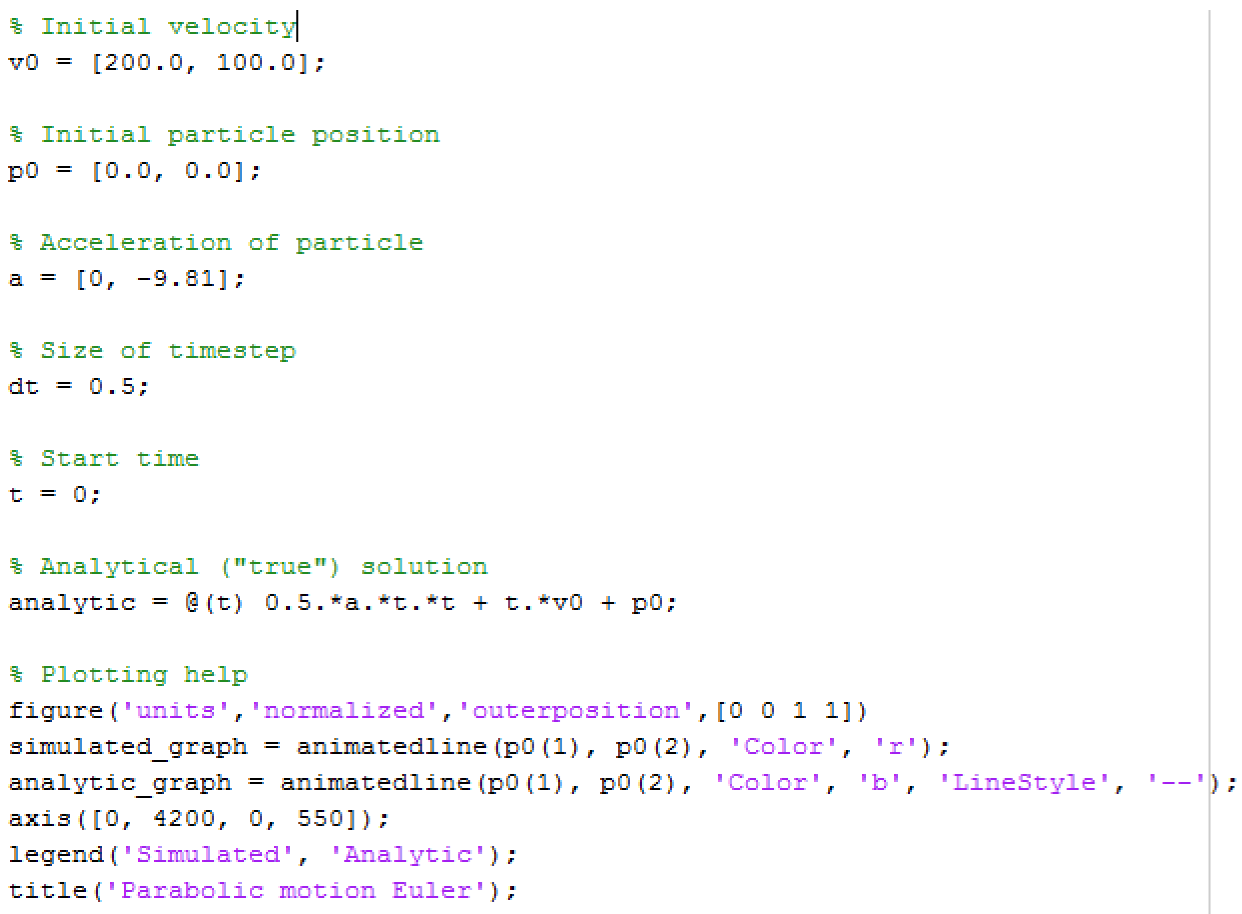
$$p^{n+1} = \frac{1}{2}(p^n + p^{**})$$

- In summary, we need to think about how we discretize our problem!

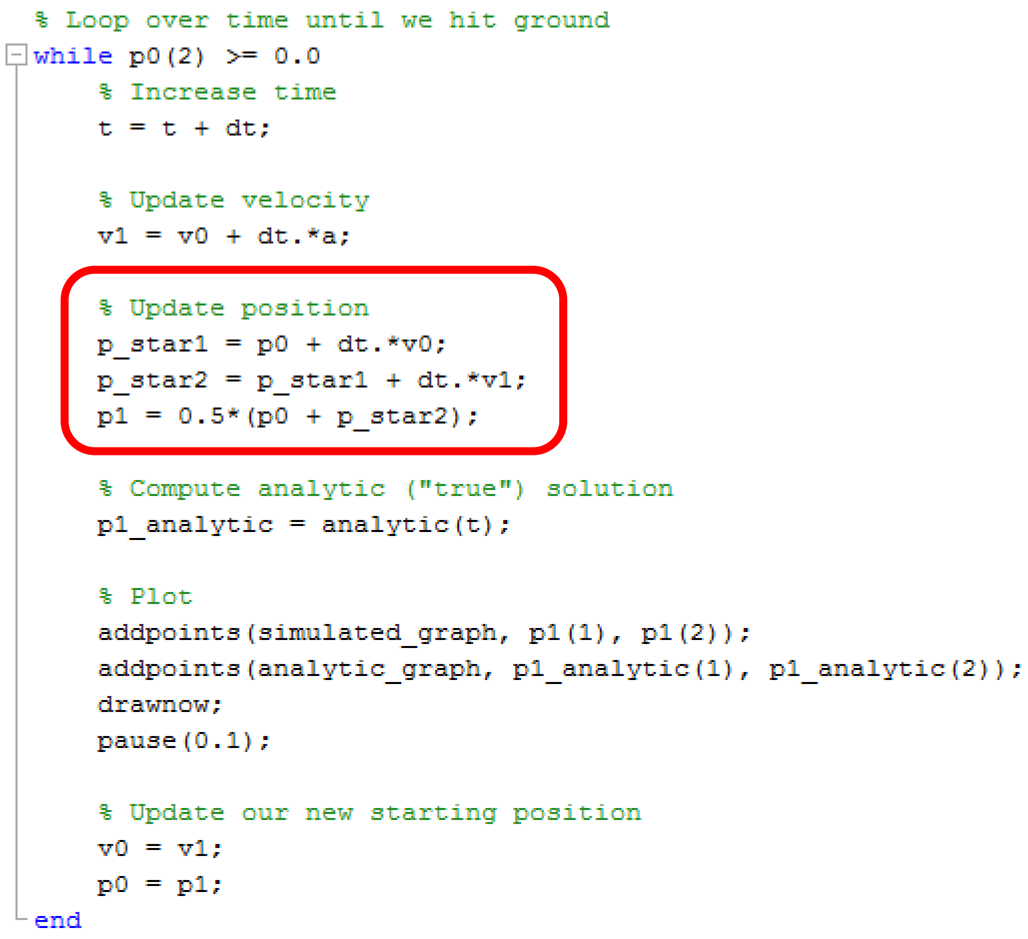
dt=1



Particle projectory in Matlab



```
% Initial velocity
v0 = [200.0, 100.0];

% Initial particle position
p0 = [0.0, 0.0];

% Acceleration of particle
a = [0, -9.81];

% Size of timestep
dt = 0.5;

% Start time
t = 0;

% Analytical ("true") solution
analytic = @(t) 0.5.*a.*t.*t + t.*v0 + p0;

% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = animatedline(p0(1), p0(2), 'Color', 'r');
analytic_graph = animatedline(p0(1), p0(2), 'Color', 'b', 'LineStyle', '--');
axis([0, 4200, 0, 550]);
legend('Simulated', 'Analytic');
title('Parabolic motion Euler');
```



```
% Loop over time until we hit ground
while p0(2) >= 0.0
    % Increase time
    t = t + dt;

    % Update velocity
    v1 = v0 + dt.*a;

    % Update position
    p_star1 = p0 + dt.*v0;
    p_star2 = p_star1 + dt.*v1;
    p1 = 0.5*(p0 + p_star2);

    % Compute analytic ("true") solution
    p1_analytic = analytic(t);

    % Plot
    addpoints(simulated_graph, p1(1), p1(2));
    addpoints(analytic_graph, p1_analytic(1), p1_analytic(2));
    drawnow;
    pause(0.1);

    % Update our new starting position
    v0 = v1;
    p0 = p1;
end
```

Partial Differential Equations (PDEs)

- Many natural phenomena can (partly) be described mathematically as conservation laws
  - Magneto-hydrodynamics
  - Traffic jams
  - Shallow water
  - Groundwater flow
  - Tsunamis
  - Sound waves
  - Heat propagation
  - Pressure waves
  - …

"Magnificent CME Erupts on the Sun - August 31" by NASA Goddard Space Flight Center - Flickr: Magnificent CME Erupts on the Sun - August 31. Licensed under CC BY 2.0 via Wikimedia Commons

Partial Differential Equations (PDEs)

- Partial differential equations (PDEs) are much like ordinary differential equations (ODEs)

- They consist of derivatives, but in this case partial derivatives.

- Partial derivatives are derivatives with respect to *one* variable
  - Example:

$$f(x, y) = x \cdot y^2$$

$$\frac{\partial f(x, y)}{\partial x} = y^2$$

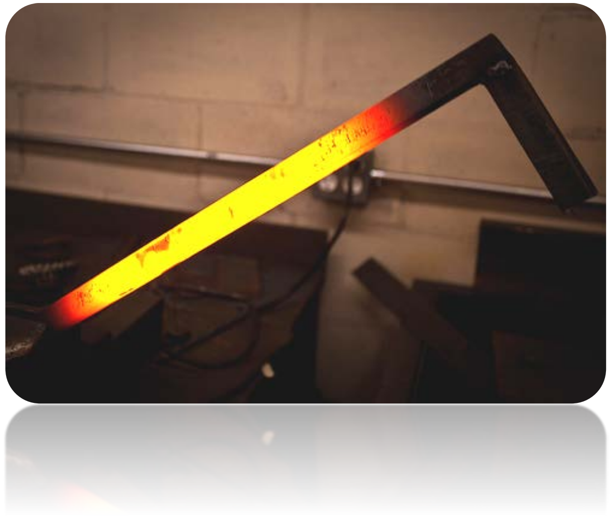
$$\frac{\partial f(x, y)}{\partial y} = 2 \cdot x \cdot y$$

- These are often impossible to solve analytically, and we must discretize them and solve on a computer.

The Heat Equation

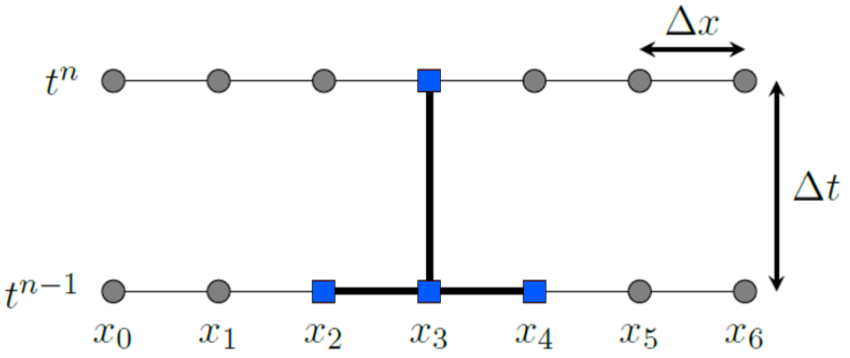
- The heat equation is a prototypical PDE (partial differential equation)

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$



- u is the temperature, kappa is the diffusion coefficient, t is time, and x is space.
- It states that the rate of change in temperature over time is equal the second derivative of the temperature with respect to space multiplied by the heat diffusion coefficient

Discretizing the heat equation

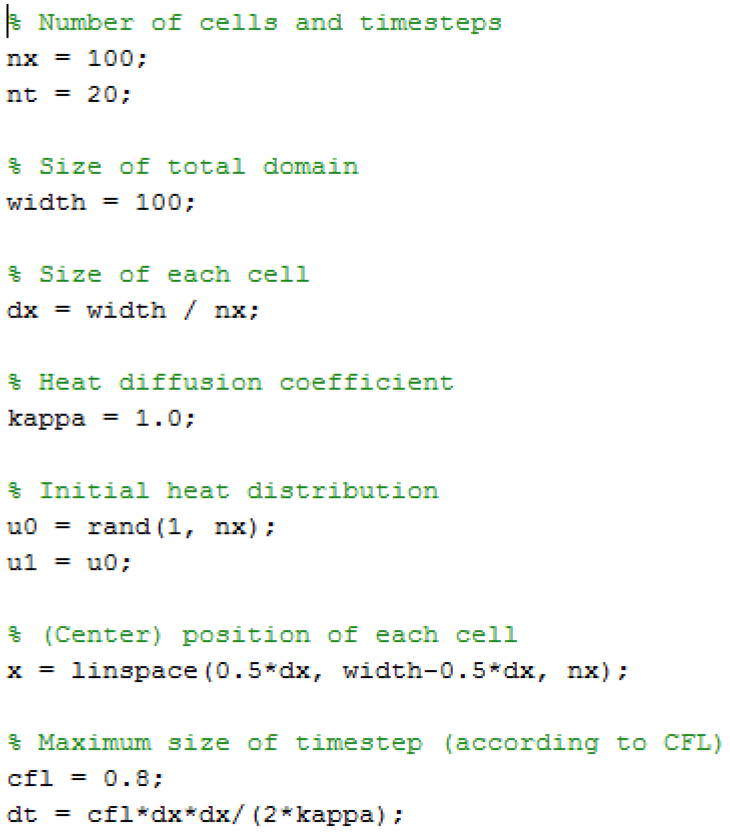


$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

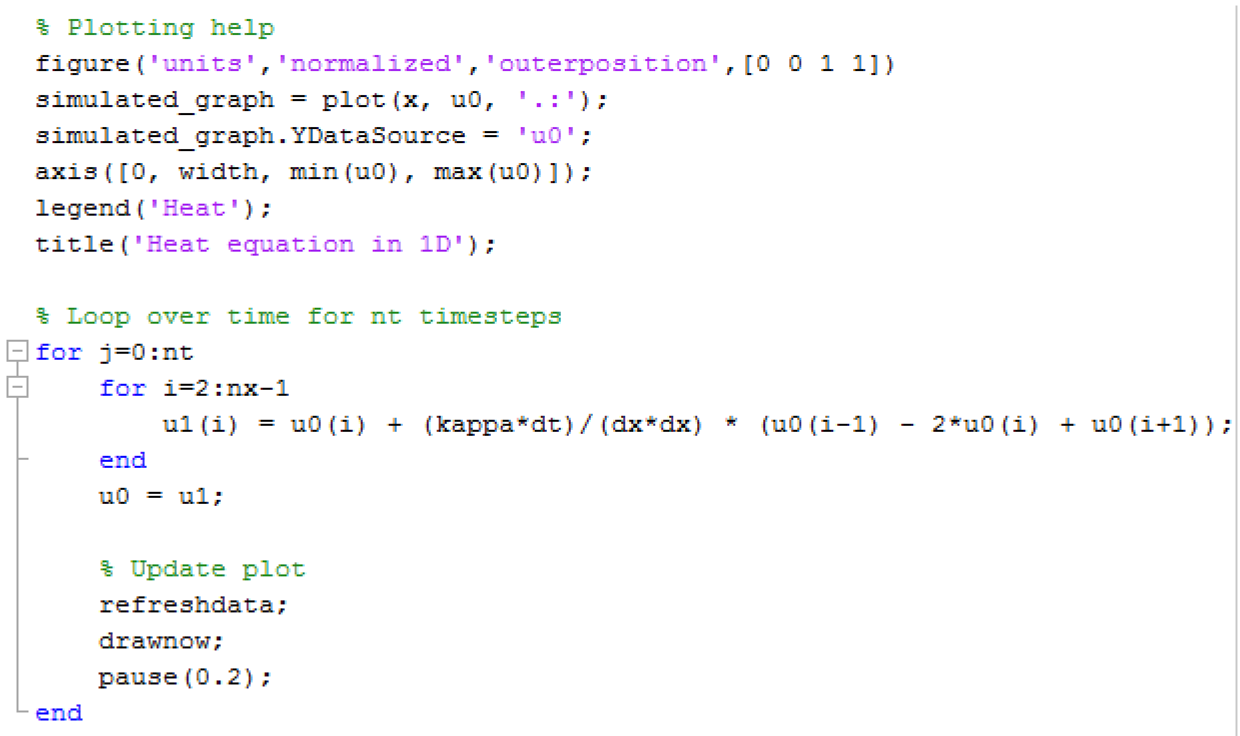
$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

$$u_i^{n+1} = ru_{i-1}^n + (1-2r)u_i^n + ru_{i+1}^n$$

The 1D heat equation in Matlab



```
% Number of cells and timesteps
nx = 100;
nt = 20;

% Size of total domain
width = 100;

% Size of each cell
dx = width / nx;

% Heat diffusion coefficient
kappa = 1.0;

% Initial heat distribution
u0 = rand(1, nx);
u1 = u0;

% (Center) position of each cell
x = linspace(0.5*dx, width-0.5*dx, nx);

% Maximum size of timestep (according to CFL)
cfl = 0.8;
dt = cfl*dx*dx/(2*kappa);
```



```
% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = plot(x, u0, '.:');
simulated_graph.YDataSource = 'u0';
axis([0, width, min(u0), max(u0)]);
legend('Heat');
title('Heat equation in 1D');

% Loop over time for nt timesteps
for j=0:nt
    for i=2:nx-1
        u1(i) = u0(i) + (kappa*dt)/(dx*dx) * (u0(i-1) - 2*u0(i) + u0(i+1));
    end
    u0 = u1;

    % Update plot
    refreshdata;
    drawnow;
    pause(0.2);
end
```

The linear wave equation in 1D

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

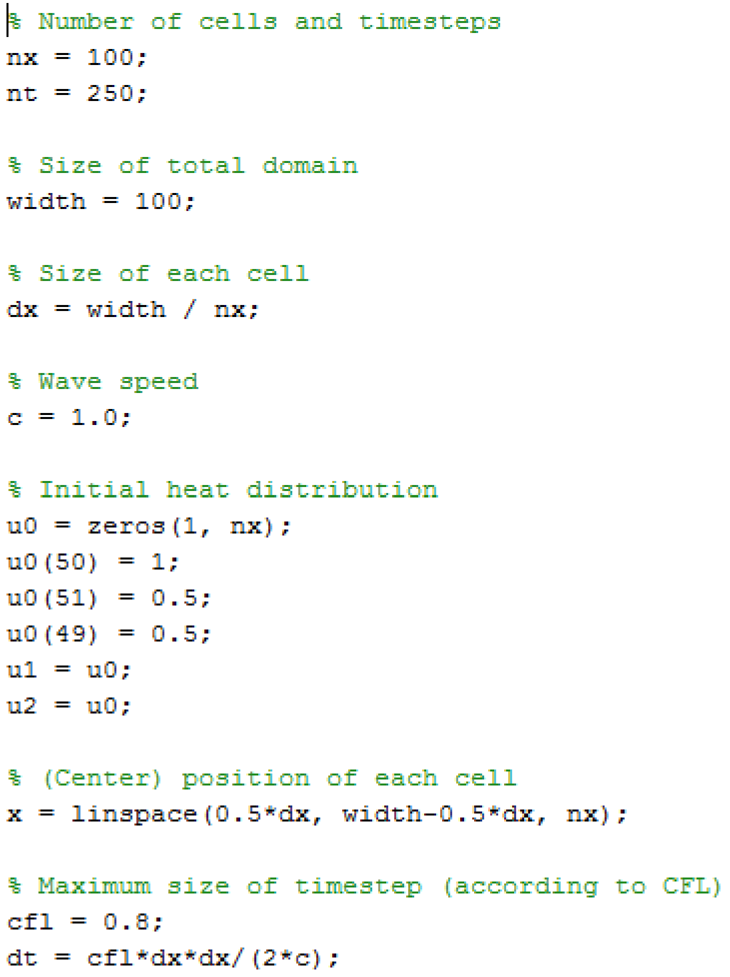
$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

$$\frac{\partial^2 u}{\partial t^2} = c\,\frac{\partial^2 u}{\partial x^2}$$

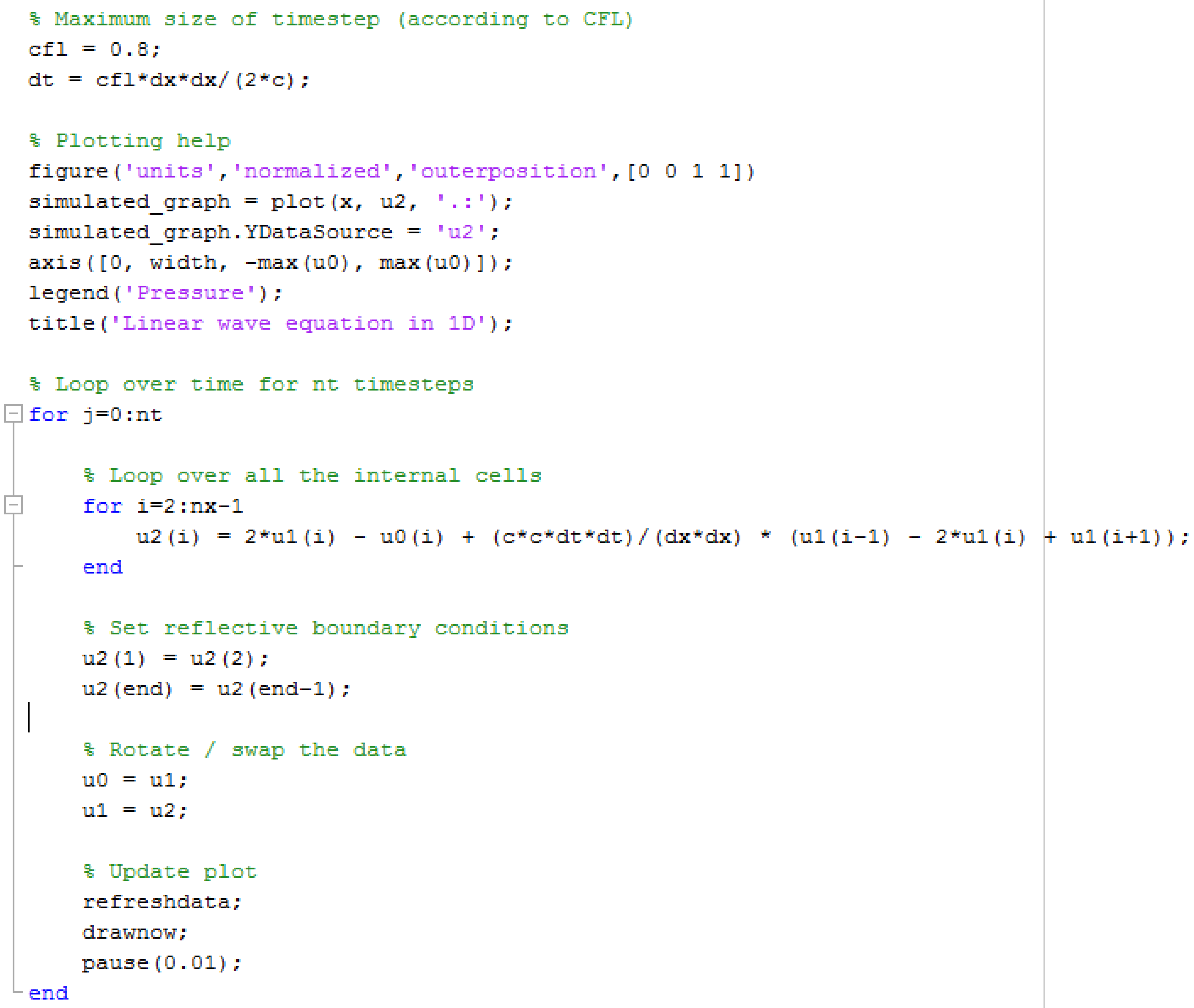
$$\frac{1}{\Delta t^2}(u_{i,j}{}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}) = \frac{c}{\Delta x^2}(u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n)$$

Soundwave photo from http://www.pngmart.com/image/34039

The 1D wave equation in Matlab



```matlab
% Number of cells and timesteps
nx = 100;
nt = 250;

% Size of total domain
width = 100;

% Size of each cell
dx = width / nx;

% Wave speed
c = 1.0;

% Initial heat distribution
u0 = zeros(1, nx);
u0(50) = 1;
u0(51) = 0.5;
u0(49) = 0.5;
u1 = u0;
u2 = u0;

% (Center) position of each cell
x = linspace(0.5*dx, width-0.5*dx, nx);

% Maximum size of timestep (according to CFL)
cfl = 0.8;
dt = cfl*dx*dx/(2*c);
```



```matlab
% Maximum size of timestep (according to CFL)
cfl = 0.8;
dt = cfl*dx*dx/(2*c);

% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = plot(x, u2, '.:');
simulated_graph.YDataSource = 'u2';
axis([0, width, -max(u0), max(u0)]);
legend('Pressure');
title('Linear wave equation in 1D');

% Loop over time for nt timesteps
for j=0:nt

    % Loop over all the internal cells
    for i=2:nx-1
        u2(i) = 2*u1(i) - u0(i) + (c*c*dt*dt)/(dx*dx) * (u1(i-1) - 2*u1(i) + u1(i+1));
    end

    % Set reflective boundary conditions
    u2(1) = u2(2);
    u2(end) = u2(end-1);

    % Rotate / swap the data
    u0 = u1;
    u1 = u2;

    % Update plot
    refreshdata;
    drawnow;
    pause(0.01);
end
```

The 2D wave equation

$$\frac{\partial^2 u}{\partial t^2} = c\,\frac{\partial^2 u}{\partial x^2}$$

$$\frac{1}{\Delta t^2}\left(u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}\right) = \frac{c}{\Delta x^2}\left(u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}\right)$$

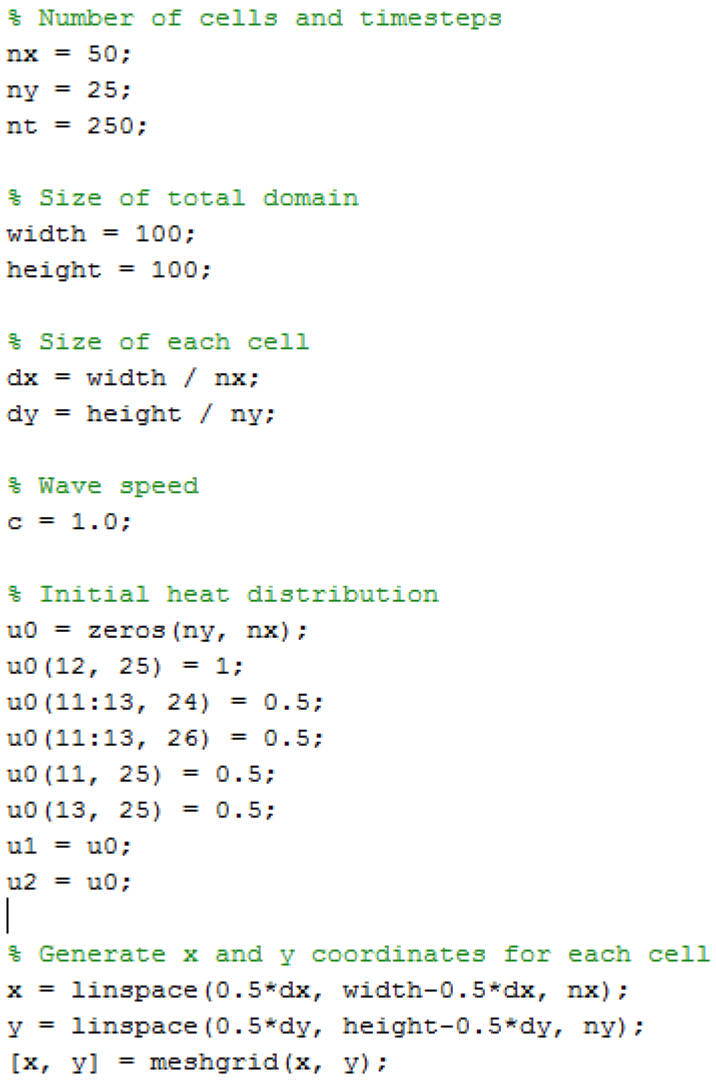
$$\frac{\partial^2 u}{\partial t^2} = c\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right]$$

$$\frac{1}{\Delta t^2}\left(u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}\right) = \frac{c}{\Delta x^2}\left(u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}\right) + \frac{c}{\Delta y^2}\left(u_{i,j-1}^{n} - 2u_{i,j}^{n} + u_{i,j+1}^{n}\right)$$

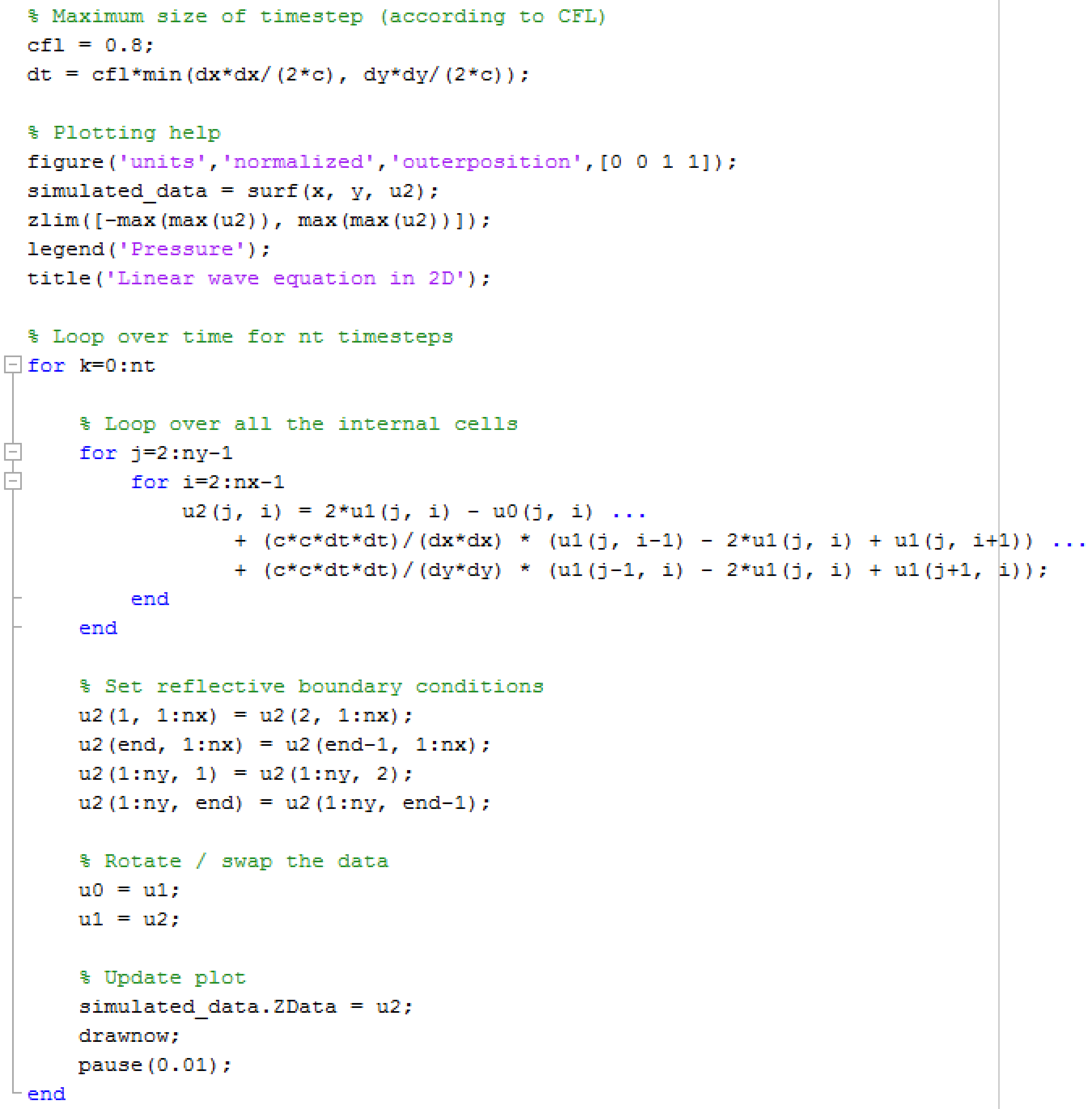
Soundwave photo from http://www.pngmart.com/image/34039

Høy C, ©Bård Breivik/BONO. Foto: Terje Heiestad. UiO

The 2D wave equation in Matlab



```matlab
% Number of cells and timesteps
nx = 50;
ny = 25;
nt = 250;

% Size of total domain
width = 100;
height = 100;

% Size of each cell
dx = width / nx;
dy = height / ny;

% Wave speed
c = 1.0;

% Initial heat distribution
u0 = zeros(ny, nx);
u0(12, 25) = 1;
u0(11:13, 24) = 0.5;
u0(11:13, 26) = 0.5;
u0(11, 25) = 0.5;
u0(13, 25) = 0.5;
u1 = u0;
u2 = u0;

% Generate x and y coordinates for each cell
x = linspace(0.5*dx, width-0.5*dx, nx);
y = linspace(0.5*dy, height-0.5*dy, ny);
[x, y] = meshgrid(x, y);
```

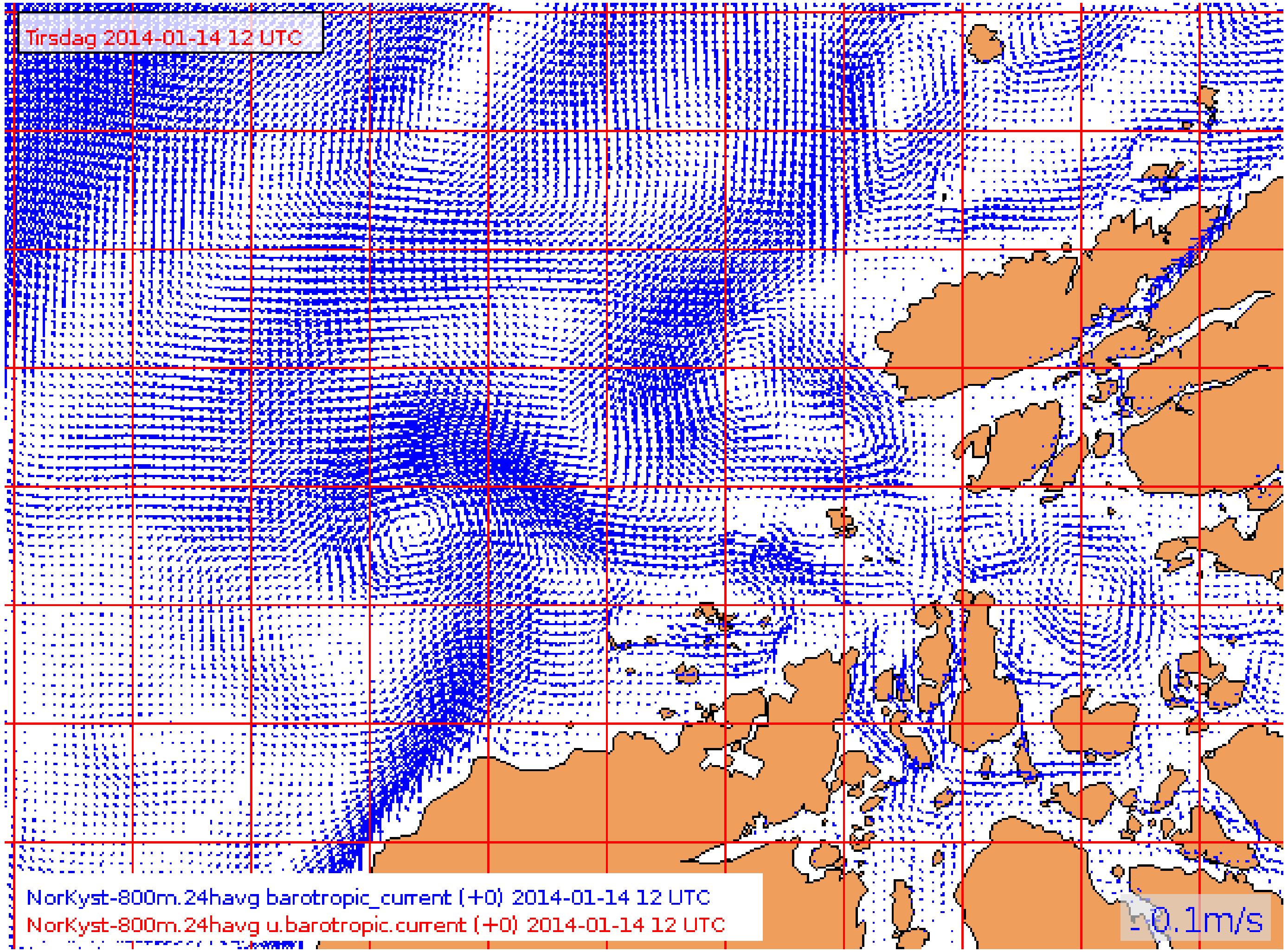


```matlab
% Maximum size of timestep (according to CFL)
cfl = 0.8;
dt = cfl*min(dx*dx/(2*c), dy*dy/(2*c));

% Plotting help
figure('units','normalized','outerposition',[0 0 1 1]);
simulated_data = surf(x, y, u2);
zlim([-max(max(u2)), max(max(u2))]);
legend('Pressure');
title('Linear wave equation in 2D');

% Loop over time for nt timesteps
for k=0:nt

    % Loop over all the internal cells
    for j=2:ny-1
        for i=2:nx-1
            u2(j, i) = 2*u1(j, i) - u0(j, i) ...
                + (c*c*dt*dt)/(dx*dx) * (u1(j, i-1) - 2*u1(j, i) + u1(j, i+1)) ...
                + (c*c*dt*dt)/(dy*dy) * (u1(j-1, i) - 2*u1(j, i) + u1(j+1, i));
        end
    end

    % Set reflective boundary conditions
    u2(1, 1:nx) = u2(2, 1:nx);
    u2(end, 1:nx) = u2(end-1, 1:nx);
    u2(1:ny, 1) = u2(1:ny, 2);
    u2(1:ny, end) = u2(1:ny, end-1);

    % Rotate / swap the data
    u0 = u1;
    u1 = u2;

    % Update plot
    simulated_data.ZData = u2;
    drawnow;
    pause(0.01);
end
```

Using conservation laws in real life

Th

Problem statement



Solution strategy

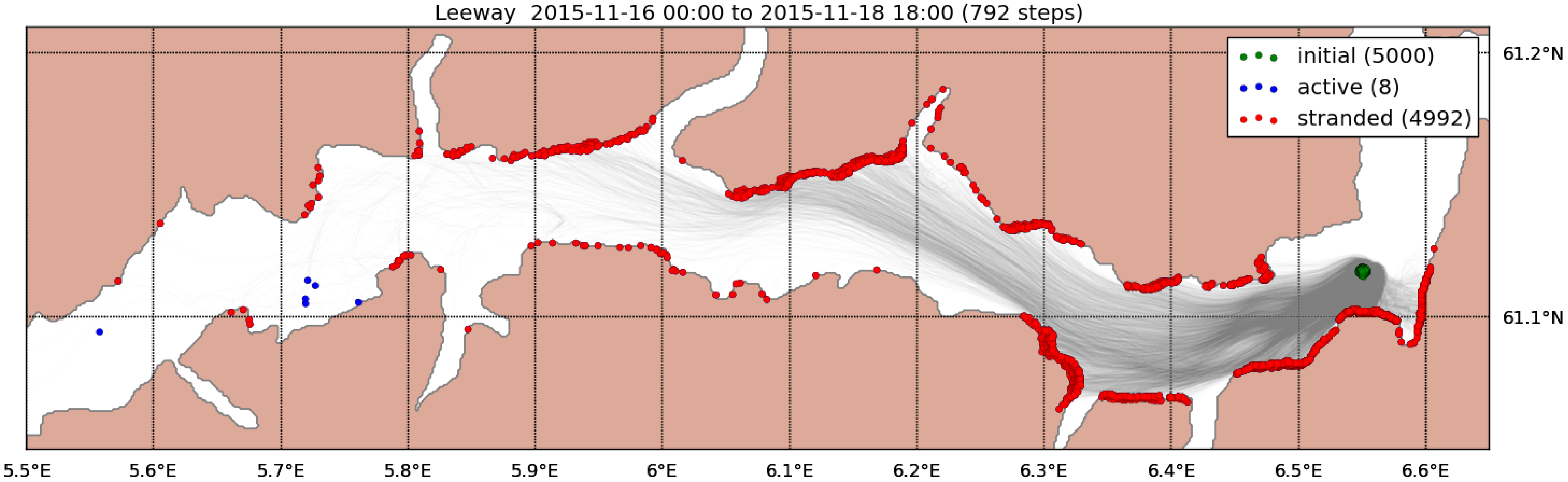
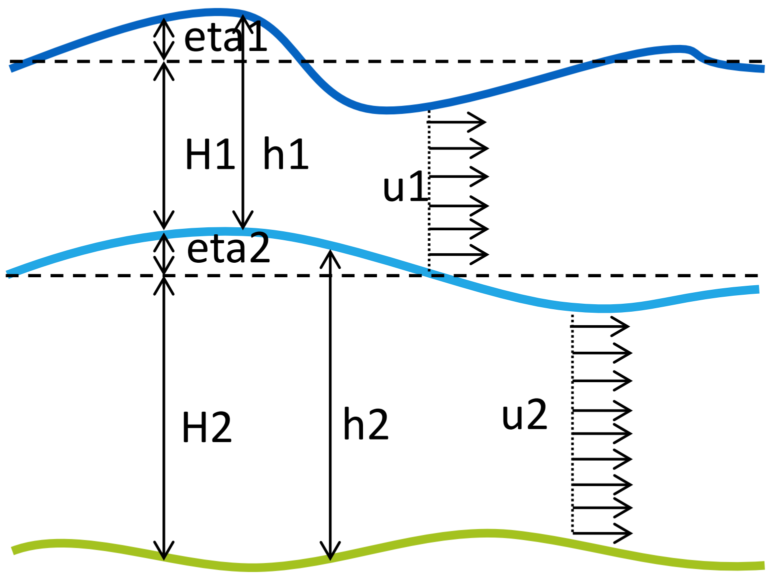


Image from Opendrift, Norwegian Meteorological Institute, Knut-Frode Dagestad.

2-layer non-linear scheme



- 1 layer model extendible to more layers
  - Ocean can be modeled as a stratisfied medium with multiple homogeneous layers
- Multiple layers enables baroclinic response from model

1 layer scheme, non-linear FD

$$\eta_{jk}^{n+1} = \eta_{jk}^{n-1} - \frac{2\Delta t}{\Delta x}\left(U_{jk}^{n} - U_{j-1k}^{n}\right) - \frac{2\Delta t}{\Delta y}\left(V_{jk}^{n} - V_{jk-1}^{n}\right),$$

$$V_{jk}^{n+1} = \frac{1}{C_{jk}^{y}}\left[V_{jk}^{n-1} + 2\Delta t\left(-f\overline{U}_{jk}^{n} + \frac{N_{jk}^{y}}{\Delta y} + \frac{P_{jk}^{y} + \hat{P}_{jk}^{y}}{\Delta y} + Y_{jk}^{n+1} + AE_{jk}^{y}\right)\right]$$

$$C_{jk}^{y} = 1 + \frac{2R\Delta t}{H_{jk}^{y}} + \frac{2A\Delta t(\Delta x^2 + \Delta y^2)}{\Delta x^2 \Delta y^2},$$

$$\begin{aligned} N_{jk}^{y} &= \frac{1}{4}\left\{\frac{\left(V_{jk+1}^{n} + V_{jk}^{n}\right)^2}{H_{jk+1}^{n} + \eta_{jk+1}^{n}} - \frac{\left(V_{jk}^{n} + V_{jk-1}^{n}\right)^2}{H_{jk}^{n} + \eta_{jk}^{n}}\right. \\ &+ \left.\frac{\Delta y}{\Delta x}\left[\frac{(U_{jk+1}^{n} + U_{jk}^{n})(V_{j+1k}^{n} + V_{jk}^{n})}{\overline{H}_{jk}^{n} + \overline{\eta}_{jk}^{n}} - \frac{(U_{j-1k+1}^{n} + U_{j-1k}^{n})(V_{jk}^{n} + V_{j-1k}^{n})}{\overline{H}_{jk-1}^{n} + \overline{\eta}_{jk-1}^{n}}\right]\right\}, \end{aligned}$$

$$P_{jk}^{y} = gH_{jk}^{y}\left(\eta_{jk+1}^{n} - \eta_{jk}^{n}\right), \quad \hat{P}_{jk}^{y} = \frac{1}{2}\left[(\eta_{jk+1}^{n})^2 - (\eta_{jk}^{n})^2\right],$$

$$E_{jk}^{y} = \frac{1}{\Delta x^2}\left(V_{j+1k}^{n} - V_{jk}^{n-1} + V_{j-1k}^{n}\right) + \frac{1}{\Delta y^2}\left(V_{jk+1}^{n} - V_{jk}^{n-1} + V_{jk-1}^{n}\right).$$

$$U_{jk}^{n+1} = \frac{1}{C_{jk}^{x}}\left[U_{jk}^{n-1} + 2\Delta t\left(f\overline{V}_{jk}^{n} + \frac{N_{jk}^{x}}{\Delta x} + \frac{P_{jk}^{x} + \hat{P}_{jk}^{x}}{\Delta x} + X_{jk}^{n+1} + AE_{jk}^{x}\right)\right],$$

$$C_{jk}^{x} = 1 + \frac{2R\Delta t}{H_{jk}^{x}} + \frac{2A\Delta t(\Delta x^2 + \Delta y^2)}{\Delta x^2 \Delta y^2},$$

$$\begin{aligned} N_{jk}^{x} &= \frac{1}{4}\left\{\frac{\left(U_{j+1k}^{n} + U_{jk}^{n}\right)^2}{H_{j+1k} + \eta_{j+1k}^{n}} - \frac{\left(U_{jk}^{n} + U_{j-1k}^{n}\right)^2}{H_{jk} + \eta_{jk}^{n}}\right. \\ &+ \left.\frac{\Delta x}{\Delta y}\left[\frac{(U_{jk+1}^{n} + U_{jk}^{n})(V_{j+1k}^{n} + V_{jk}^{n})}{\overline{H}_{jk}^{n} + \overline{\eta}_{jk}^{n}} - \frac{(U_{jk}^{n} + U_{jk-1}^{n})(V_{j+1k-1}^{n} + V_{jk-1}^{n})}{\overline{H}_{jk-1}^{n} + \overline{\eta}_{jk-1}^{n}}\right]\right\}, \end{aligned} \quad (23)$$

$$P_{jk}^{x} = gH_{jk}^{x}\left(\eta_{j+1k}^{n} - \eta_{jk}^{n}\right), \quad \hat{P}_{jk}^{x} = \frac{1}{2}\left[(\eta_{j+1k}^{n})^2 - (\eta_{jk}^{n})^2\right],$$

$$E_{jk}^{x} = \frac{1}{\Delta x^2}\left(U_{j+1k}^{n} - U_{jk}^{n-1} + U_{j-1k}^{n}\right) + \frac{1}{\Delta y^2}\left(U_{jk+1}^{n} - U_{jk}^{n-1} + U_{jk-1}^{n}\right),$$

Oppsummering

Oppsummering

- Matte er gøy :D!
- Ting virker ofte mye vanskeligere enn de er: konseptene er ofte enkle
- Har man forstått konseptene så kommer detaljene på plass
- Sterk kunnskap i både matte og data er viktig for effektiv problemløsing

Oppgaver

- Ta utgangspunkt i utlevert kildekode (ikke løsningsforslag!)
- Implementer i følgende rekkefølge
  - ParabolicMotionEuler.m
  - ParabolicMotionRK2.m
  - HeatEquation1D.m
  - WaveEquation1D.m
  - WaveEquation2D.m

- Hvis du blir ferdig:
  - Hvordan kan du gjøre disse operasjonene mer effektive?
  - Implementer HeatEquation2D.m (uten skjellettkode)

Hjelp til oppgaver

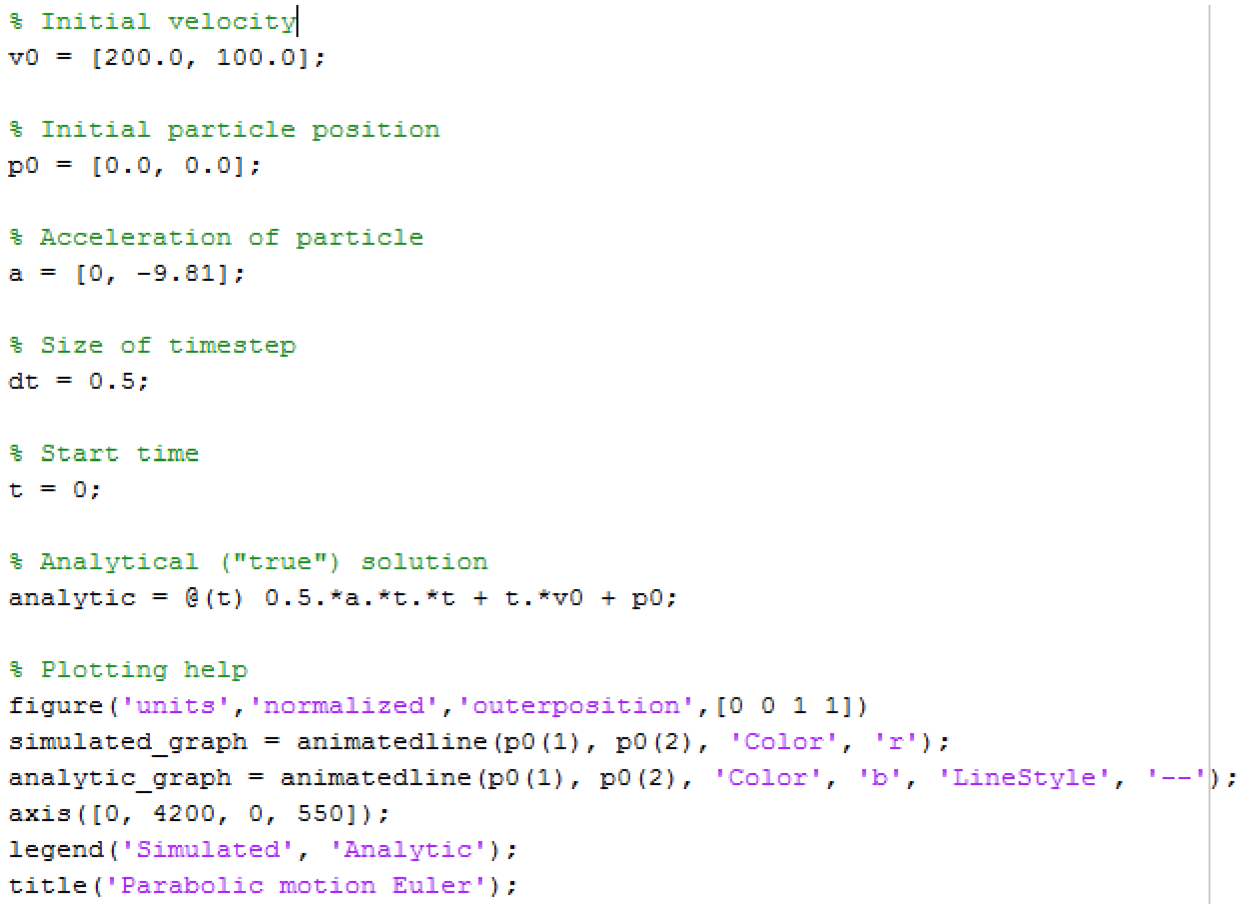
Parabolic motion (Euler)

$$\frac{d\vec{p}}{dt} = \vec{v}(t)$$

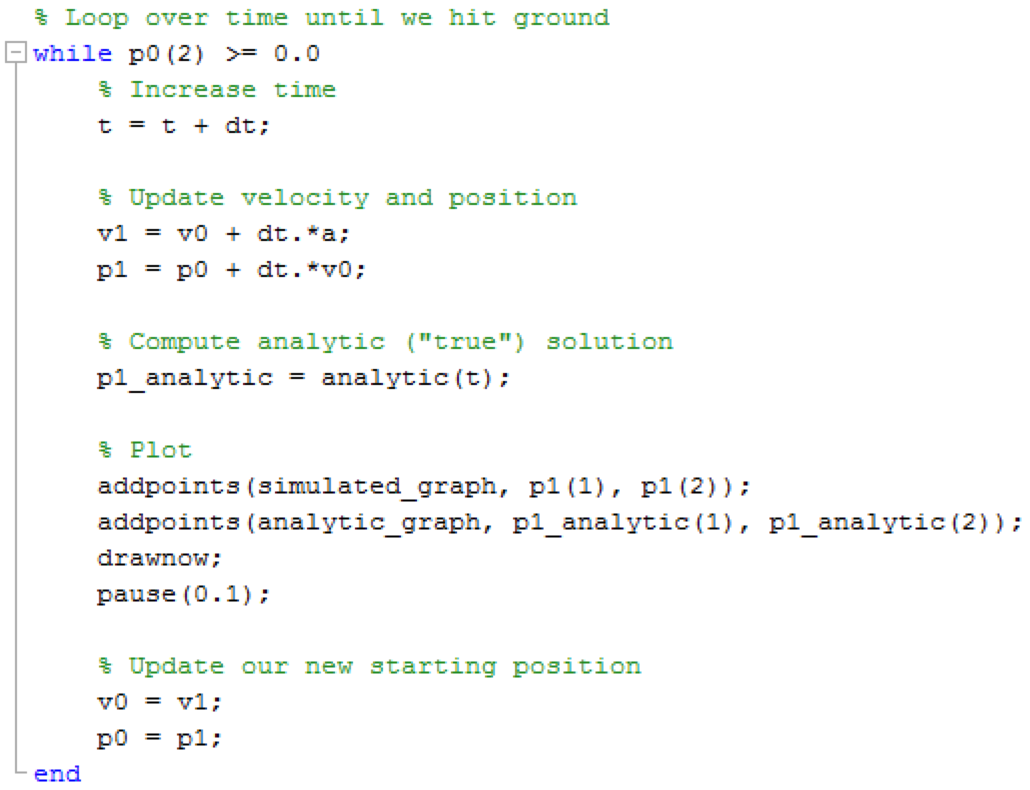
$$\frac{\vec{p}^{n+1} - \vec{p}^{n}}{\Delta t} = \vec{v}(n \cdot \Delta t)$$

$$\vec{p}^{n+1} = \vec{p}^{n} + \Delta t \cdot \vec{v}(n \cdot \Delta t)$$

Particle projectory in Matlab



```
% Initial velocity
v0 = [200.0, 100.0];

% Initial particle position
p0 = [0.0, 0.0];

% Acceleration of particle
a = [0, -9.81];

% Size of timestep
dt = 0.5;

% Start time
t = 0;

% Analytical ("true") solution
analytic = @(t) 0.5.*a.*t.*t + t.*v0 + p0;

% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = animatedline(p0(1), p0(2), 'Color', 'r');
analytic_graph = animatedline(p0(1), p0(2), 'Color', 'b', 'LineStyle', '--');
axis([0, 4200, 0, 550]);
legend('Simulated', 'Analytic');
title('Parabolic motion Euler');
```



```
% Loop over time until we hit ground
while p0(2) >= 0.0
    % Increase time
    t = t + dt;

    % Update velocity and position
    v1 = v0 + dt.*a;
    p1 = p0 + dt.*v0;

    % Compute analytic ("true") solution
    p1_analytic = analytic(t);

    % Plot
    addpoints(simulated_graph, p1(1), p1(2));
    addpoints(analytic_graph, p1_analytic(1), p1_analytic(2));
    drawnow;
    pause(0.1);

    % Update our new starting position
    v0 = v1;
    p0 = p1;
end
```

More accuracy

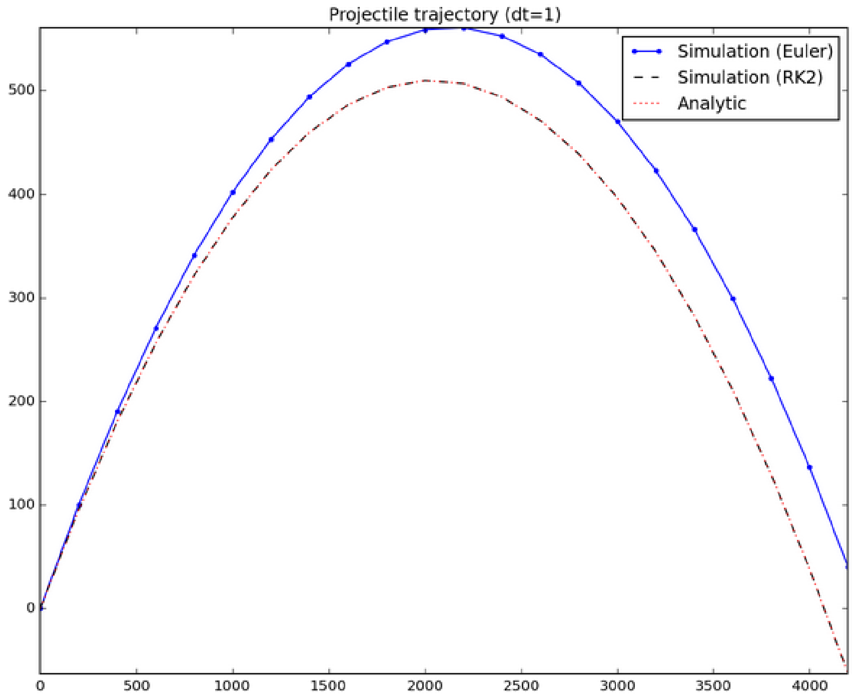
- We have used a very simple integration rule (or approximation to the derivative)
  - Our rule is known as forward Euler

$$p^{n+1} = p^n + \Delta t \cdot \vec{v}$$

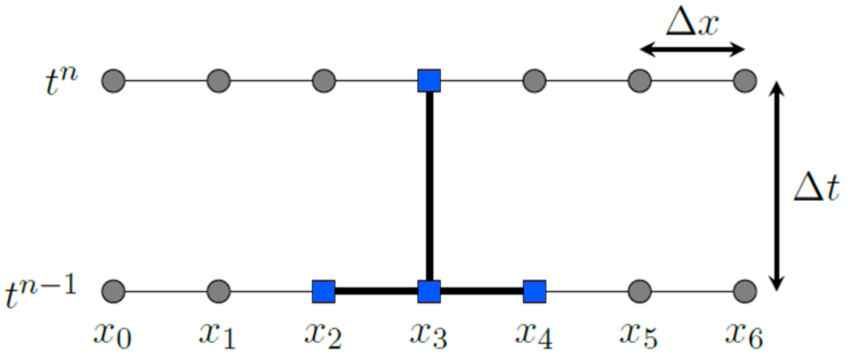
- We can get much higher accuracy with more advanced techniques such as Runge-Kutta 2

$$p^* = p^n + \Delta t \cdot \vec{v}(n \cdot \Delta t)$$
$$p^{**} = p^* + \Delta t \cdot \vec{v}\big((n+1) \cdot \Delta t\big)$$
$$p^{n+1} = \frac{1}{2}(p^n + p^{**})$$

- In summary, we need to think about how we discretize our problem!


dt=1

Discretizing the heat equation

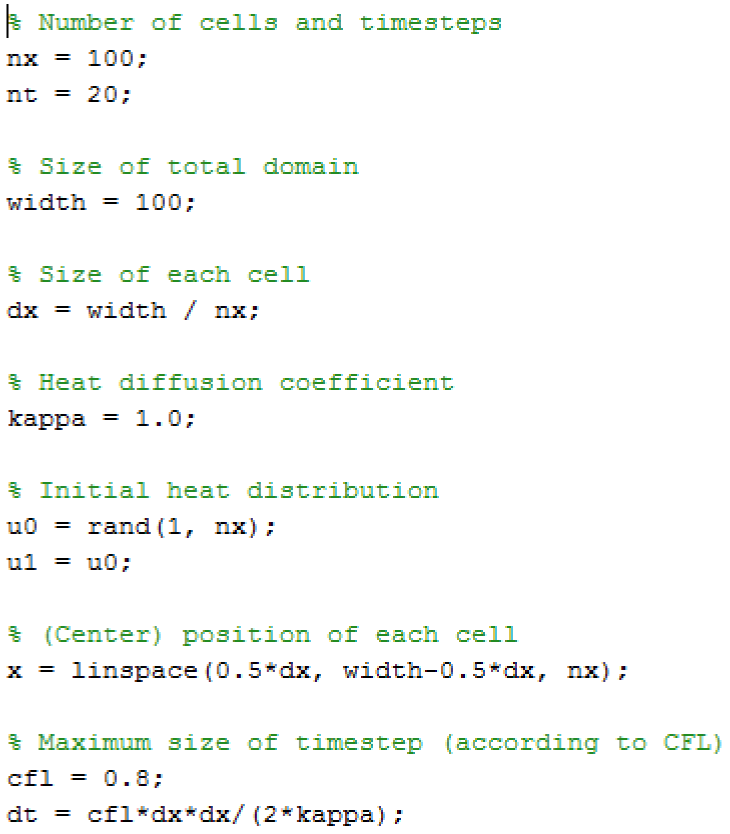


$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

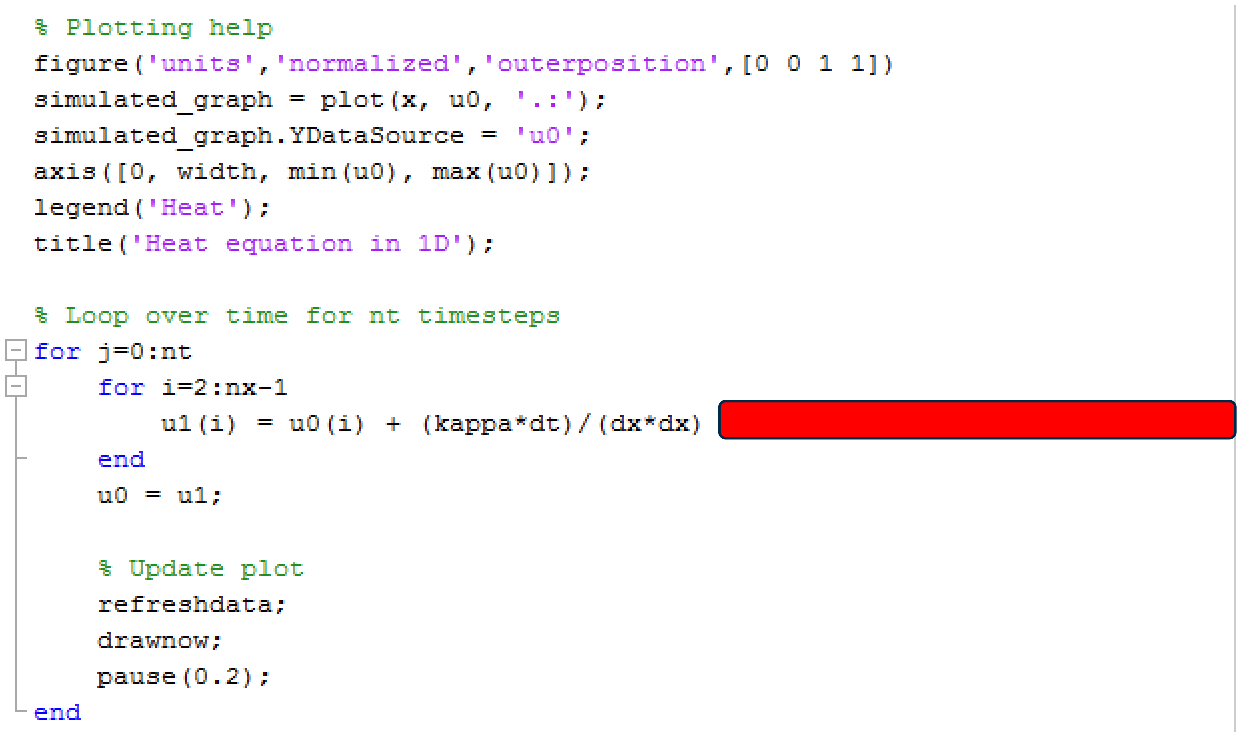
$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

$$u_i^{n+1} = ru_{i-1}^n + (1-2r)u_i^n + ru_{i+1}^n$$

The 1D heat equation in Matlab

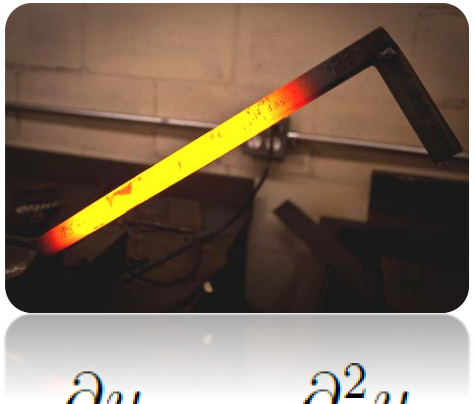


```
% Number of cells and timesteps
nx = 100;
nt = 20;

% Size of total domain
width = 100;

% Size of each cell
dx = width / nx;

% Heat diffusion coefficient
kappa = 1.0;

% Initial heat distribution
u0 = rand(1, nx);
u1 = u0;

% (Center) position of each cell
x = linspace(0.5*dx, width-0.5*dx, nx);

% Maximum size of timestep (according to CFL)
cfl = 0.8;
dt = cfl*dx*dx/(2*kappa);
```



```
% Plotting help
figure('units','normalized','outerposition',[0 0 1 1])
simulated_graph = plot(x, u0, '.:');
simulated_graph.YDataSource = 'u0';
axis([0, width, min(u0), max(u0)]);
legend('Heat');
title('Heat equation in 1D');

% Loop over time for nt timesteps
for j=0:nt
    for i=2:nx-1
        u1(i) = u0(i) + (kappa*dt)/(dx*dx) [redacted]
    end
    u0 = u1;

    % Update plot
    refreshdata;
    drawnow;
    pause(0.2);
end
```

The linear wave equation in 1D



$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

$$\frac{\partial^2 u}{\partial t^2} = c\,\frac{\partial^2 u}{\partial x^2}$$

$$\frac{1}{\Delta t^2}({u_{i,j}}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}) = \frac{c}{\Delta x^2}(u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n)$$

Soundwave photo from http://www.pngmart.com/image/34039

The 2D wave equation

$$\frac{\partial^2 u}{\partial t^2} = c\,\frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial^2 u}{\partial t^2} = c\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right]$$

$$\frac{1}{\Delta t^2}\left(u_{i,j}{}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}\right) = \frac{c}{\Delta x^2}\left(u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}\right)$$

$$\frac{1}{\Delta t^2}\left(u_{i,j}{}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}\right) = \frac{c}{\Delta x^2}\left(u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}\right) + \frac{c}{\Delta y^2}\left(u_{i,j-1}^{n} - 2u_{i,j}^{n} + u_{i,j+1}^{n}\right)$$

Soundwave photo from http://www.pngmart.com/image/34039

Høy C, ©Bård Breivik/BONO. Foto: Terje Heiestad. UiO