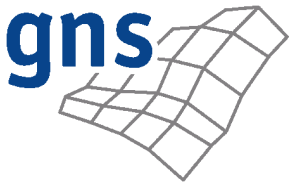
# Numerical Fractional Calculus Using Methods Based on Non-Uniform Step Sizes

Kai Diethelm

Gesellschaft für
numerische Simulation mbH
Braunschweig




AG Numerik
Institut Computational Mathematics
Technische Universität Braunschweig


International Symposium on Fractional PDEs
June 3–5, 2013

# Cooperation partners

- Neville J. Ford (University of Chester, UK)
- Alan D. Freed (Saginaw Valley State University)

SPONSORED BY THE
Federal Ministry of Education and Research

(Grant No. 01IH11006C)

# Table of Contents

# Mathematical Fundamentals

Problem: Find a numerical solution to the fractional order IVP

$$\begin{aligned} D^{\alpha}_{*0} y(t) &= f(t, y(t)) \\ y^{(k)}(0) &= y_0^{(k)} \quad (k = 0, 1, \ldots, \lceil \alpha \rceil - 1) \end{aligned}$$

for

$$t \in [0, T]$$

where

$$D^{\alpha}_{*0} = \text{Caputo differential operator of order } \alpha.$$

In this talk: $0 < \alpha \leq 1$
(generalization to $\alpha > 1$ usually no problem)

# Classical Approach

- Use uniform mesh

$$t_j = j \cdot h$$

where

$$h = \frac{T}{N}$$

with some suitably chosen parameter $N \in \mathbb{N}$,
- discretize the fractional operators,
- solve the resulting discrete problem

(Oldham & Spanier 1974, Lubich 1983ff., ...)

# Disadvantage

- Fractional differential operators are not local:

$$D^{\alpha}_{*0}y(t) = \frac{1}{\Gamma(n-\alpha)} \int_0^t (t-s)^{n-\alpha-1} D^n y(s) ds$$

  where $n = \lceil \alpha \rceil$
- Treatment of process history increases computational complexity
- Standard solution approach has $O(N^2)$ operation count
- Integer-order equations: operation count is only $O(N)$

# Example: Adams-Bashforth-Moulton Method

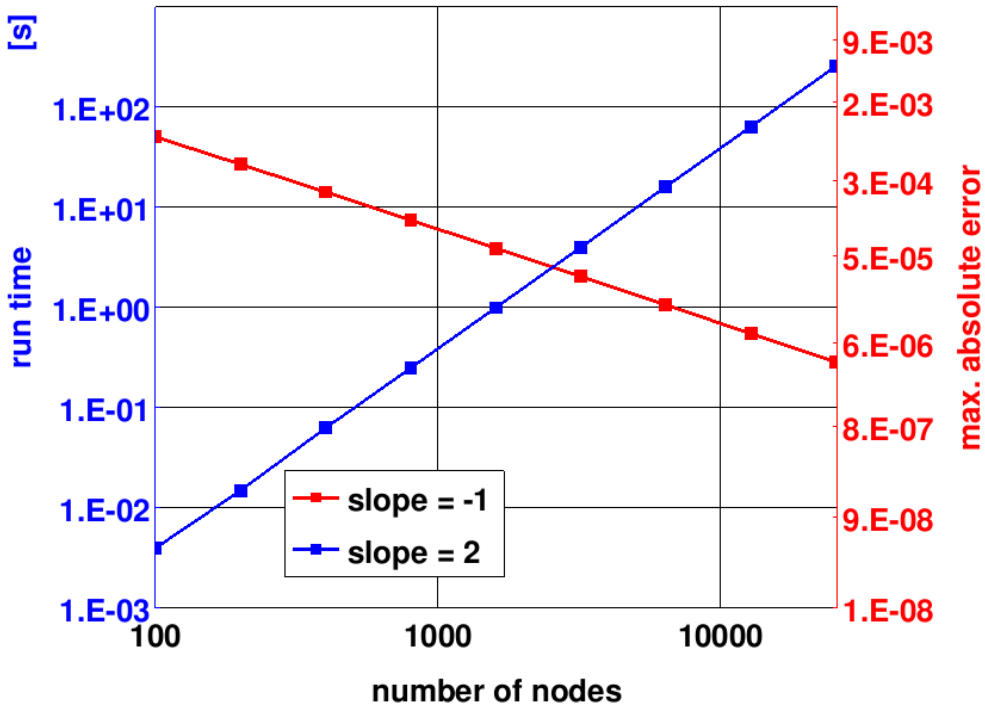
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 1: Smooth solution

$$D_{*0}^{0.4} y(t) = -y(t) + t^2 - t + \frac{2t^{1.6}}{\Gamma(2.6)} - \frac{t^{0.6}}{\Gamma(1.6)}$$

$$y(0) = 0$$

exact solution:
$y(t) = t^2 - t$



# Example: Adams-Bashforth-Moulton Method

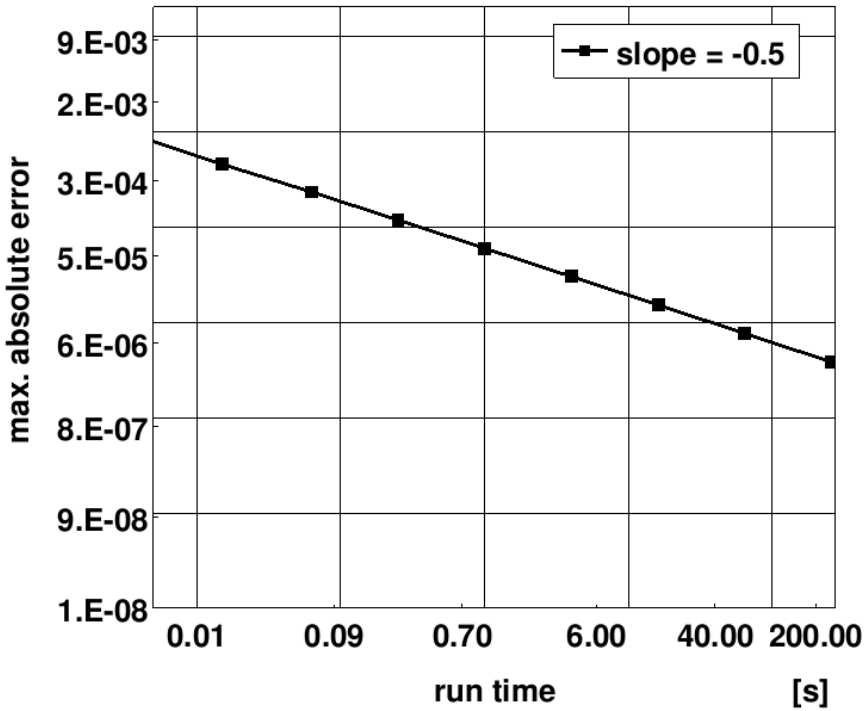
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 1: Smooth solution

$$D_{*0}^{0.4} y(t) = -y(t) + t^2 - t + \frac{2t^{1.6}}{\Gamma(2.6)} - \frac{t^{0.6}}{\Gamma(1.6)}$$

$$y(0) = 0$$

exact solution:
$y(t) = t^2 - t$



# Example: Adams-Bashforth-Moulton Method

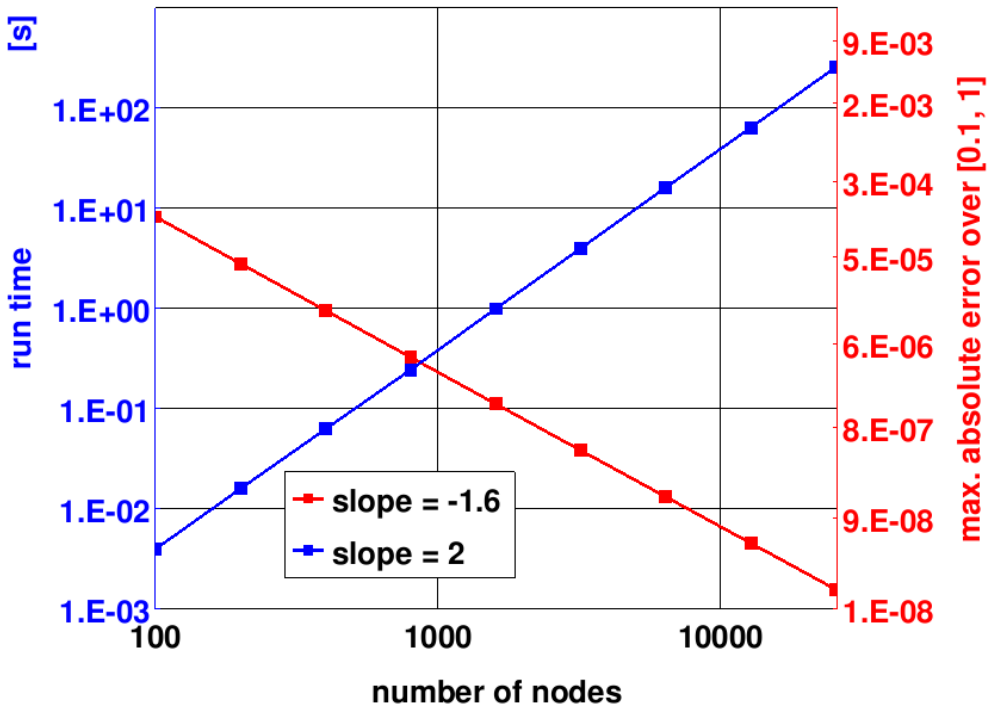
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 1: Smooth solution

$$D_{*0}^{0.4} y(t) = -y(t) + t^2 - t + \frac{2t^{1.6}}{\Gamma(2.6)} - \frac{t^{0.6}}{\Gamma(1.6)}$$

$$y(0) = 0$$

exact solution:
$y(t) = t^2 - t$



# Example: Adams-Bashforth-Moulton Method

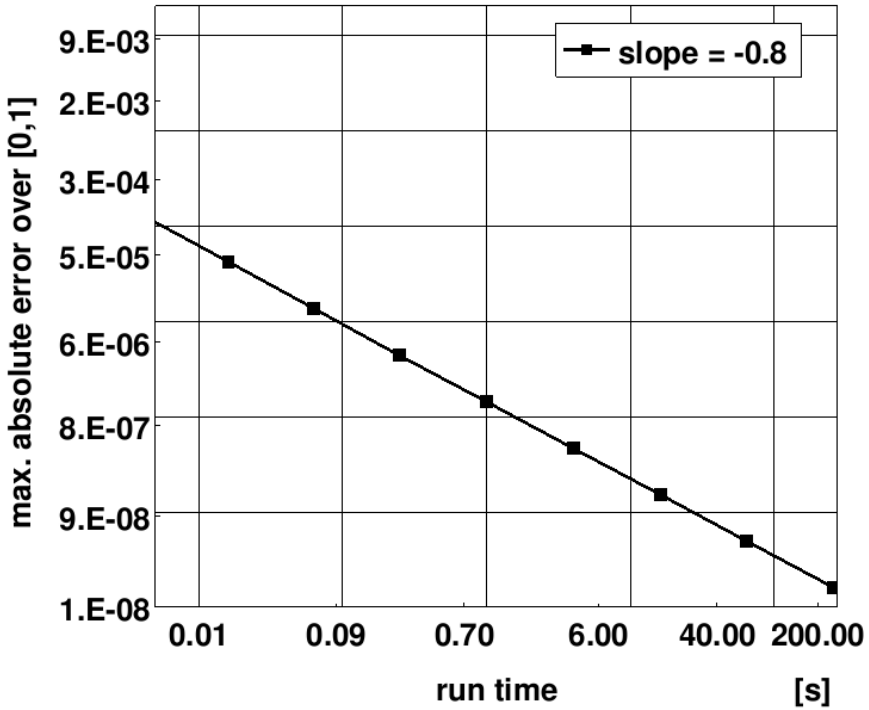
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 1: Smooth solution

$$D_{*0}^{0.4} y(t) = -y(t) + t^2 - t + \frac{2t^{1.6}}{\Gamma(2.6)} - \frac{t^{0.6}}{\Gamma(1.6)}$$

$$y(0) = 0$$

exact solution:
$y(t) = t^2 - t$



# Example: Adams-Bashforth-Moulton Method

Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

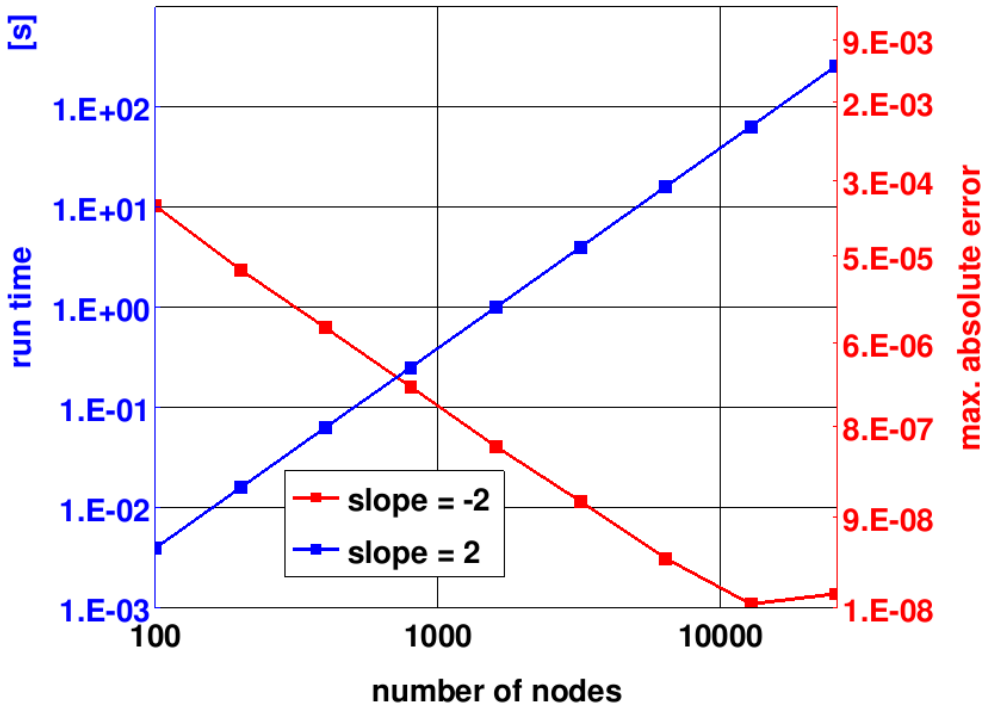
Case 2: Equation with smooth fractional derivative of solution

$$D_{*0}^{0.4} y(t) = -(y(t))^{3/2} + \frac{40320}{\Gamma(8.6)} t^{7.6} - 3\frac{\Gamma(5.2)}{\Gamma(4.8)} t^{3.8} + \frac{9}{4}\Gamma(1.4) + \left(\frac{3}{2}t^{0.2} - t^4\right)^3$$

$$y(0) = 0$$

exact solution:

$$y(t) = t^8 - 3t^{4.2} + \frac{9}{4}t^{0.4}$$



# Example: Adams-Bashforth-Moulton Method

Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

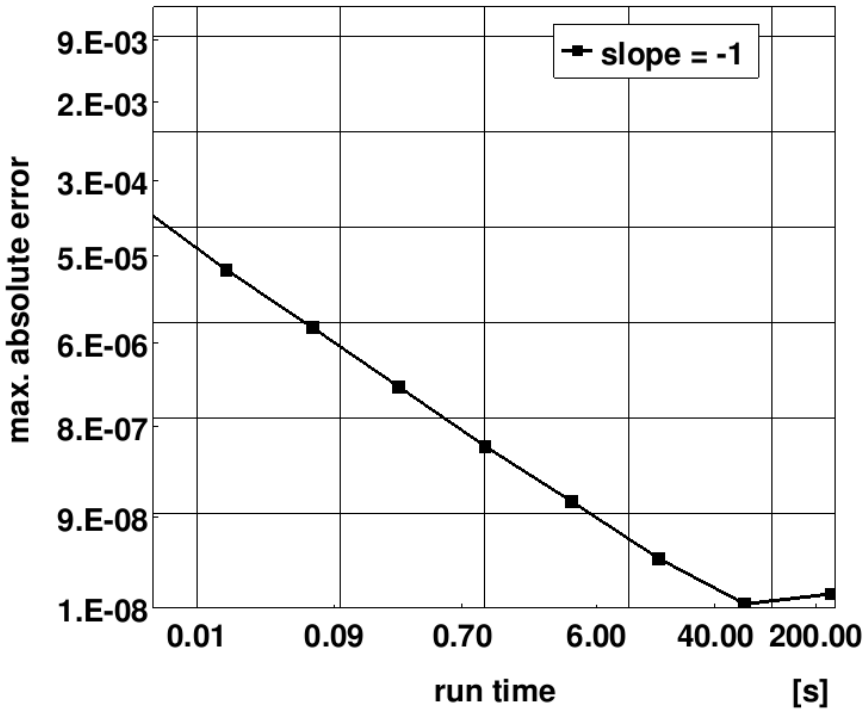
Case 2: Equation with smooth fractional derivative of solution

$$D_{*0}^{0.4} y(t) = -(y(t))^{3/2} + \frac{40320}{\Gamma(8.6)} t^{7.6} - 3\frac{\Gamma(5.2)}{\Gamma(4.8)} t^{3.8} + \frac{9}{4}\Gamma(1.4) + \left(\frac{3}{2} t^{0.2} - t^4\right)^3$$

$$y(0) = 0$$

exact solution:

$$y(t) = t^8 - 3t^{4.2} + \frac{9}{4} t^{0.4}$$



# Example: Adams-Bashforth-Moulton Method

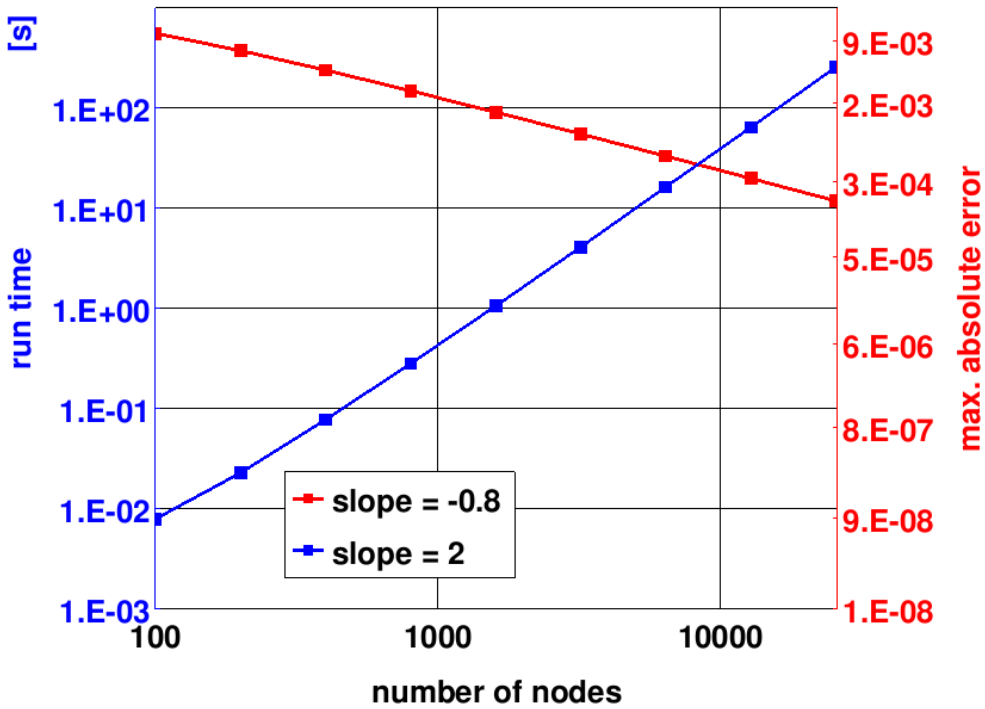
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 3: Smooth right-hand side (nonsmooth solution)

$$\begin{aligned} D_{*0}^{0.4} y(t) &= -2y(t) \\ y(0) &= 1 \end{aligned}$$

exact solution:

$$y(t) = E_{0.4}(-2x^{0.4})$$



# Example: Adams-Bashforth-Moulton Method

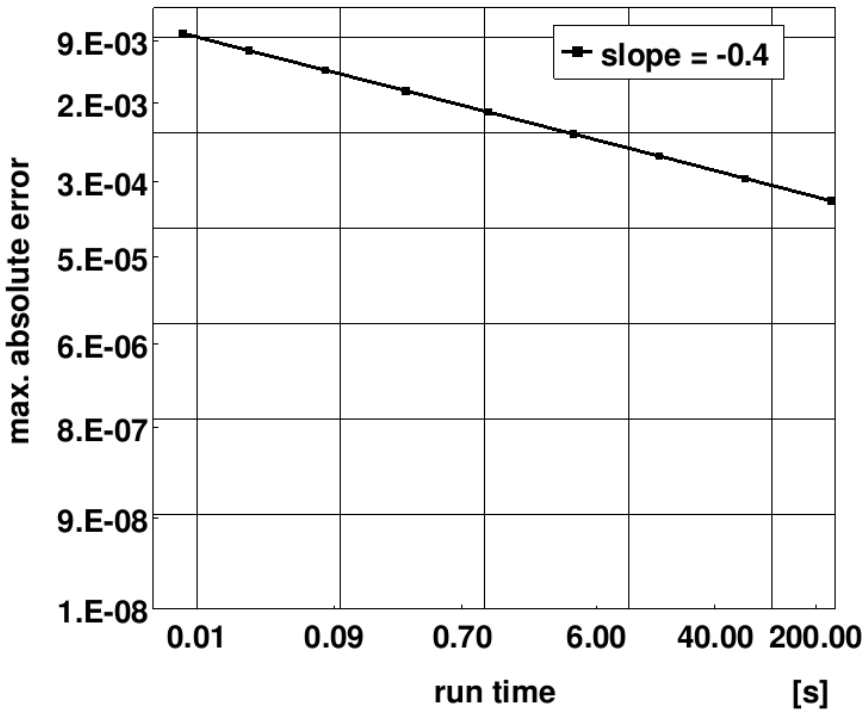
Method of P(EC)$^m$E type (Di., Ford & Freed 1999ff.), $m = \lceil 1/\alpha \rceil$

Case 3: Smooth right-hand side (nonsmooth solution)

$$\begin{aligned} D_{*0}^{0.4} y(t) &= -2y(t) \\ y(0) &= 1 \end{aligned}$$

exact solution:

$$y(t) = E_{0.4}(-2x^{0.4})$$





# Table of Contents

# High Order Methods

(Lubich 1983 ff.)

Basic idea:

Construct algorithm such that

$$\text{Error} = O(N^{-p}), \qquad p \text{ large}$$

$\Rightarrow$ High accuracy can be obtained with small $N$

$\Rightarrow$ $O(N^2)$ computational cost becomes acceptable

Main tool: Set of *starting weights* added to numerical method.

# High Order Methods

Important observation (Di., Ford, Ford, Weilbeer 2006):

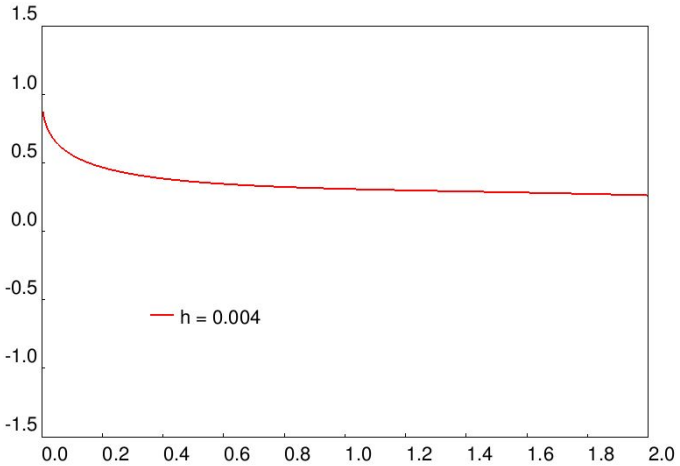
1. Starting weights are given as solutions to linear system of equations
2. Coefficient matrix is of generalized Vandermonde form
3. Depending on $\alpha$, system may be mildly ($\alpha = 0.5$) or very strongly ($\alpha = 0.4999$) ill conditioned
4. Effective computation of starting weights is potentially subject to high inaccuracies
5. Numerical results are very good in theory but possibly very poor in practice

# High Order Methods

Example: Fractional form of BDF5

$$D^{\alpha}_{*0} y(t) = -2y(t) + 0.2 \sin t, \qquad y(0) = 1$$

$$\alpha = 1/2$$

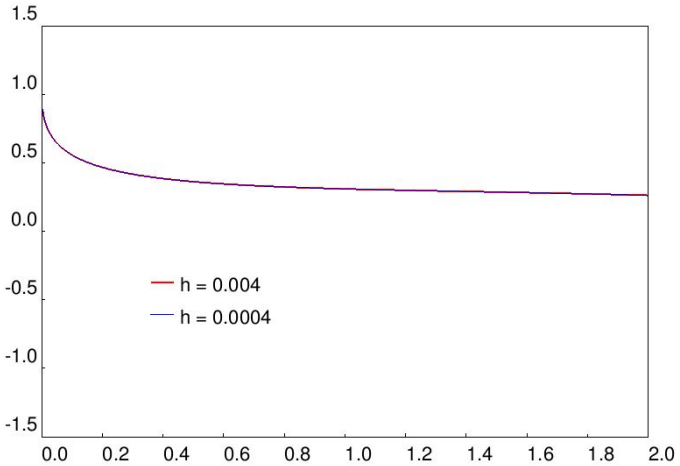


# High Order Methods

Example: Fractional form of BDF5

$$D^{\alpha}_{*0} y(t) = -2y(t) + 0.2 \sin t, \qquad y(0) = 1$$

$$\alpha = 1/2$$

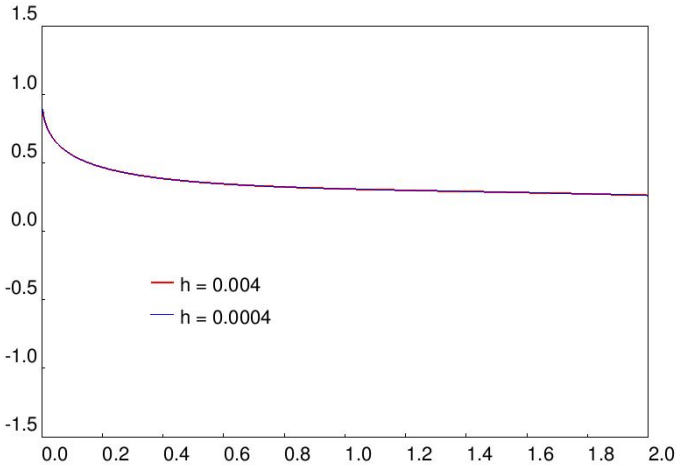


# High Order Methods

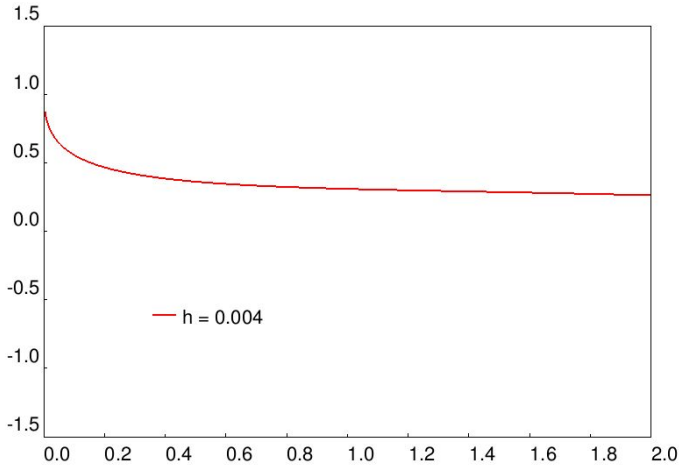
Example: Fractional form of BDF5

$$D^{\alpha}_{*0} y(t) = -2y(t) + 0.2 \sin t, \qquad y(0) = 1$$

$\alpha = 1/2$




$\alpha = 0.4999$

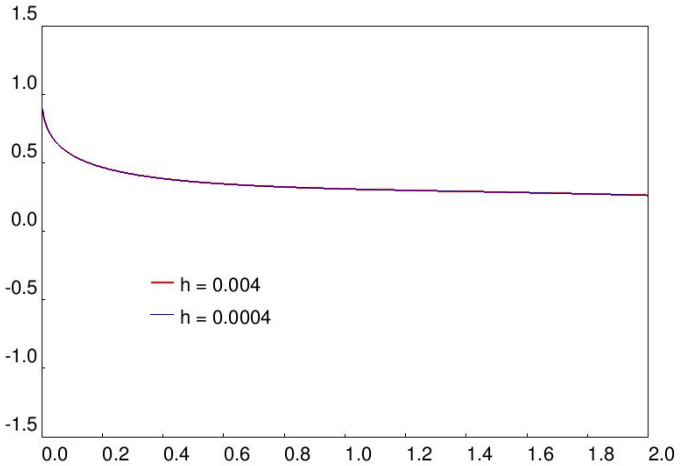


# High Order Methods

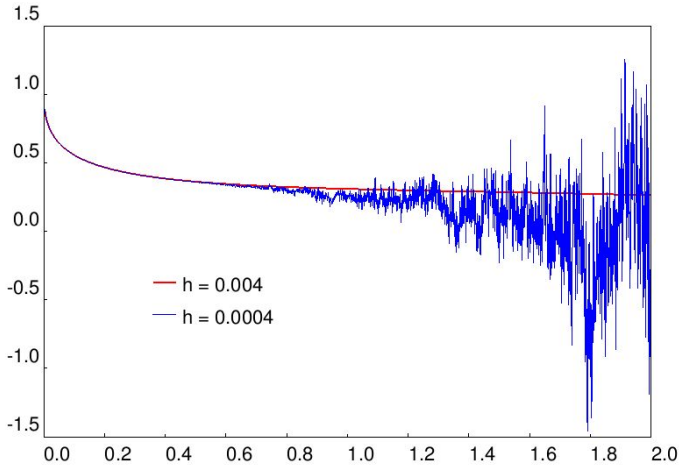
Example: Fractional form of BDF5

$$D^{\alpha}_{*0} y(t) = -2y(t) + 0.2 \sin t, \qquad y(0) = 1$$

$\alpha = 1/2$




$\alpha = 0.4999$

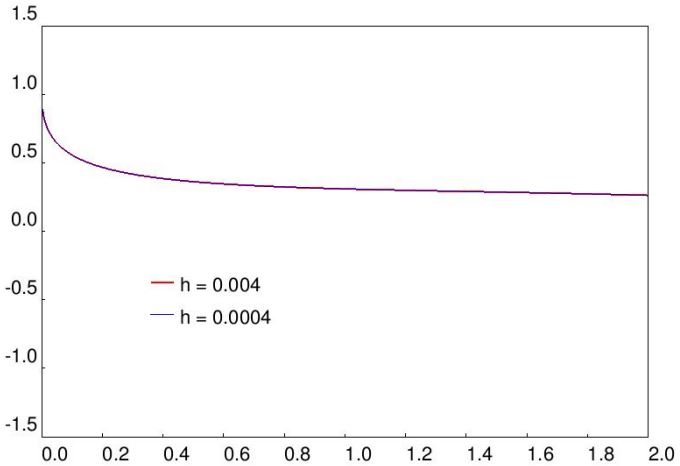


# High Order Methods

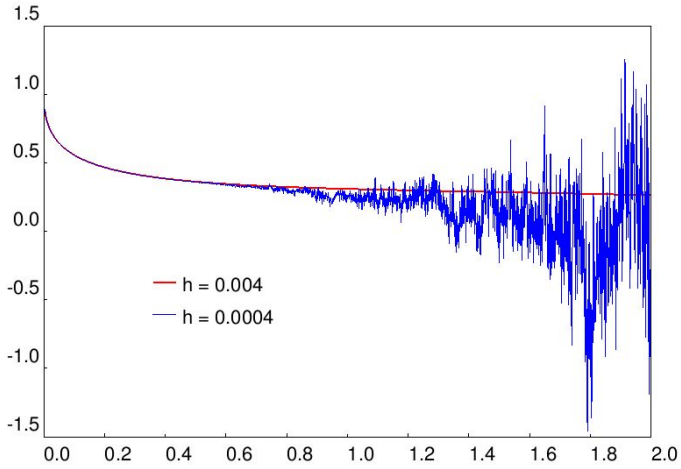
Example: Fractional form of BDF5

$$D^{\alpha}_{*0} y(t) = -2y(t) + 0.2 \sin t, \qquad y(0) = 1$$

$\alpha = 1/2$




$\alpha = 0.4999$




Conclusion: Methods works for some, but not all, $\alpha$

# Fixed Memory

(Podlubny 1999)

Rewrite given IVP in Volterra form,

$$y(t) = \sum_{k=0}^{\lceil\alpha\rceil-1} y^{(k)}(0)\frac{t^k}{k!} + \frac{1}{\Gamma(\alpha)}\int_0^t (t-s)^{\alpha-1} f(s, y(s))ds,$$

introduce parameter $\tau > 0$ (fixed memory length),

replace Volterra equation for $t > \tau$ by

$$y(t) = \sum_{k=0}^{\lceil\alpha\rceil-1} y^{(k)}(0)\frac{t^k}{k!} + \frac{1}{\Gamma(\alpha)}\int_{t-\tau}^t (t-s)^{\alpha-1} f(s, y(s))ds.$$

# Fixed Memory

Solve modified equation (with fixed memory) via numerical scheme with $O(N^{-p})$ error bound:

- Computational complexity is reduced to $O(N)$ for sufficiently small $\tau$
  (Podlubny 1999)
- Solution of original equation can be approximated with $O(N^{-p})$ accuracy only if $\tau \approx T$
  (Ford & Simpson 2001)

Approach can only yield either low computational cost or satisfactory accuracy, but not both.

# Graded Meshes

(Brunner 1985; Pedas, Tamme, Vainikko, . . . )

Basic idea:

$$\text{Improve ratio } \frac{\text{error}}{\text{run time}}$$

- by reducing the error
- without changing the run time and
- without changing the underlying approximation operator
- thus avoiding problems introduced via high order operators

# Graded Meshes

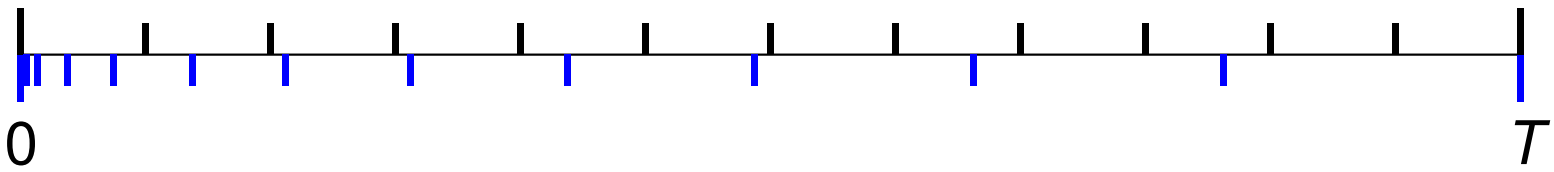
Fundamental approach:

- Reason for low convergence order:
  Poor smoothness properties of solution near the origin
- Remedy: Adapt structure of mesh to behaviour of solution
- Precise form:

$$\text{Graded mesh} \qquad t_j = \left(\frac{j}{N}\right)^{(m/\alpha)} T, \quad j = 0, 1, \ldots, N$$

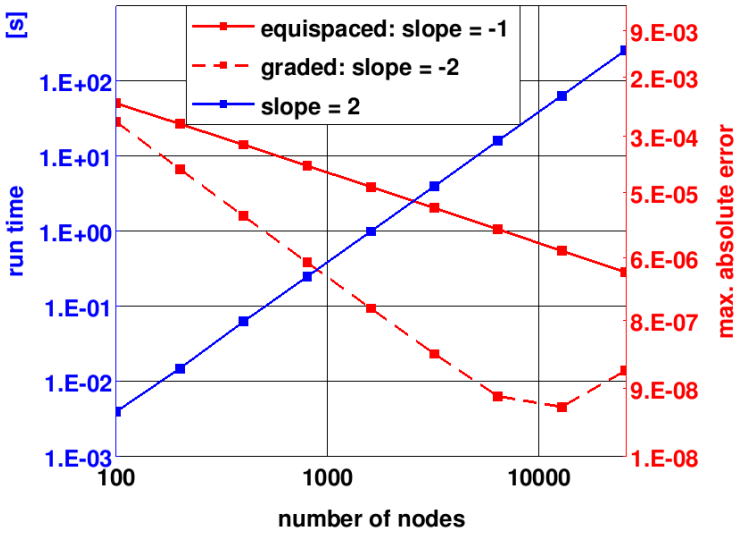
- Finer node spacing where required

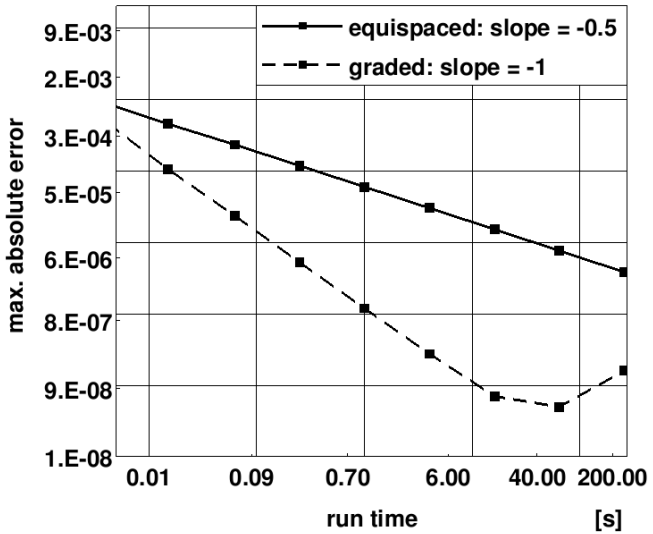
Example: uniform mesh, graded mesh ($m = 2$, $\alpha = 0.8$)



# Graded Meshes

## Example: Smooth solution (case 1 above)



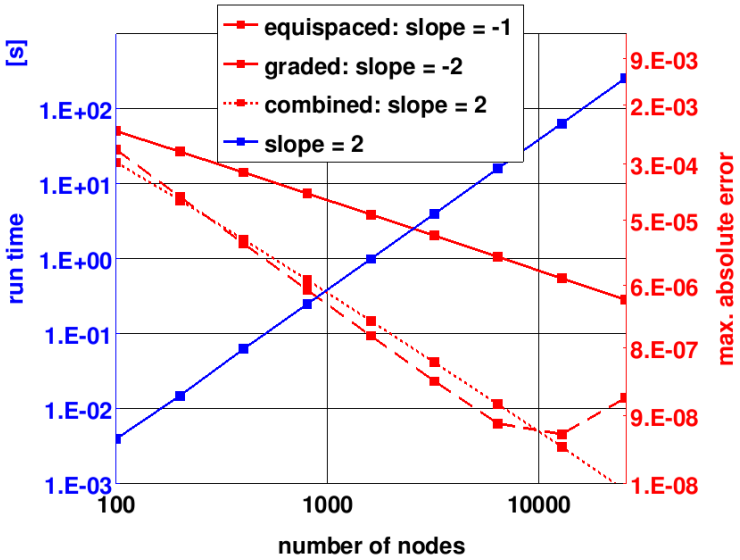

- Expected improvements achieved if $N$ is not too large
- Performance deteriorates as $N$ is increased further
- Same behaviour for other examples
- Reason: Computation of quadrature weights becomes numerically unstable (significant cancellation of digits)

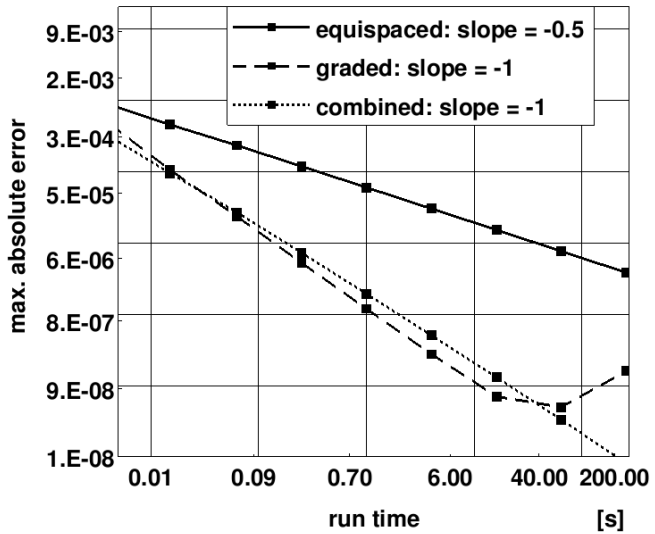
# Graded Meshes

Possible improvement:

- Subdivide interval as $[0, T] = [0, \tilde{T}] \cup [\tilde{T}, T]$
- Use graded mesh as above with $cN$ subintervals on $[0, \tilde{T}]$, where $c \ll 1$
- Use uniform mesh with $(1 - c)N$ subintervals on $[\tilde{T}, T]$

$\Rightarrow$ Problem occurs only for much larger values of $N$



# Automatic Stepsize Control

General idea:

- Mesh not defined a priori
- Start with certain step size
- Find approximation and estimate its error
- If error estimate too large then reject step and retry with smaller step size
- If error estimate very small then increase step size (reduction of computational cost)
- Otherwise continue working with present step size

$\Rightarrow$ Fine mesh used only where required by properties of solution

Open question:

- Reliable and computationally cheap estimation of error?

# A General Observation

- Two types of meshes exist in numerical scheme:
  - Find approximate solution on primary mesh $\{t_0, t_1, \ldots, t_N\}$
  - For each $j$, discretize the integral

$$\int_0^{t_j} (t_j - s)^{\alpha-1} f(s, y(s)) ds$$

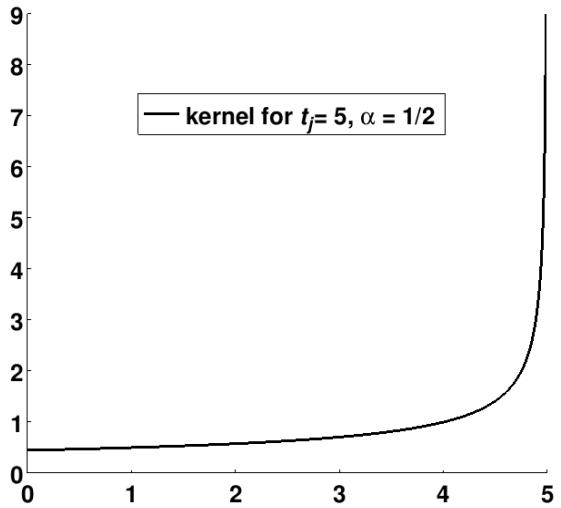
    in Volterra form of IVP on secondary mesh $\{\tau_{j,\mu} : 0 \leq \mu \leq N_j\}$
- Overall complexity $= \sum_{j=1}^{N} N_j$
- Traditional methods require that both meshes coincide:

$$N_j = j \text{ and } \tau_{j,\mu} = t_\mu \quad (\mu = 0, 1, 2, \ldots, N_j)$$

  $\Rightarrow$ complexity $= \sum_{j=1}^{N} j \approx \frac{1}{2} N^2$
- Potential improvement: use $\{\tau_{j,\mu}\}$ with much smaller $N_j$ without increasing order of associated error

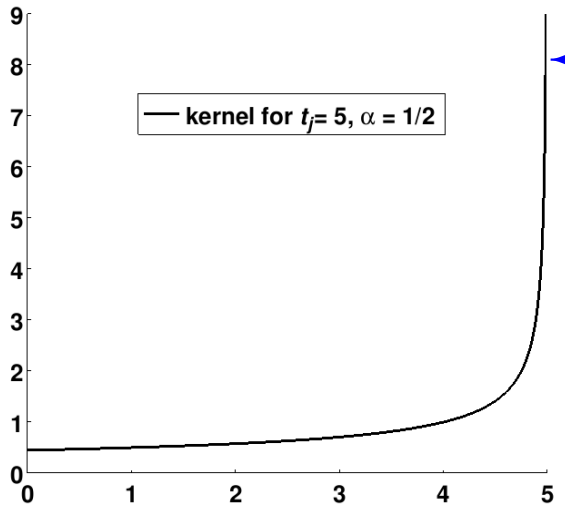
# Logarithmic Memory I

Kernel of integral operator at grid point $t_j$ is $(t_j - s)^{\alpha-1}$



# Logarithmic Memory I

Kernel of integral operator at grid point $t_j$ is $(t_j - s)^{\alpha-1}$




near $t_j$:
kernel is large

⇓

precise approximation of integrand necessary

⇓

fine secondary mesh in this region

# Logarithmic Memory I

Kernel of integral operator at grid point $t_j$ is $(t_j - s)^{\alpha-1}$

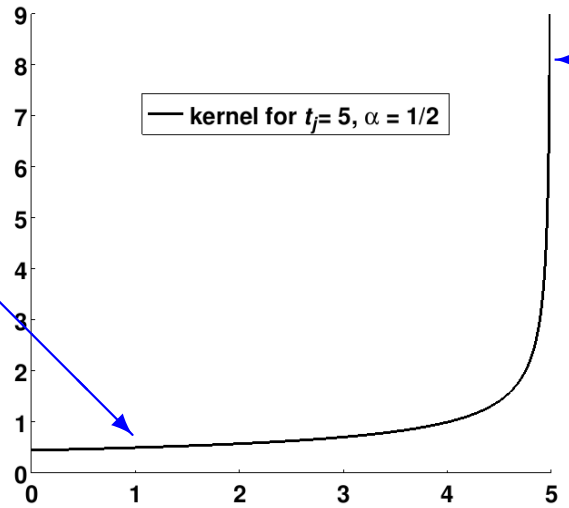
away from $t_j$:
kernel is small

$\Downarrow$

less accurate but cheap approximation of integrand suffices

$\Downarrow$

coarse secondary mesh in this region




near $t_j$:
kernel is large

$\Downarrow$

precise approximation of integrand necessary

$\Downarrow$

fine secondary mesh in this region

# Logarithmic Memory I

Given step size $h$ of primary mesh,

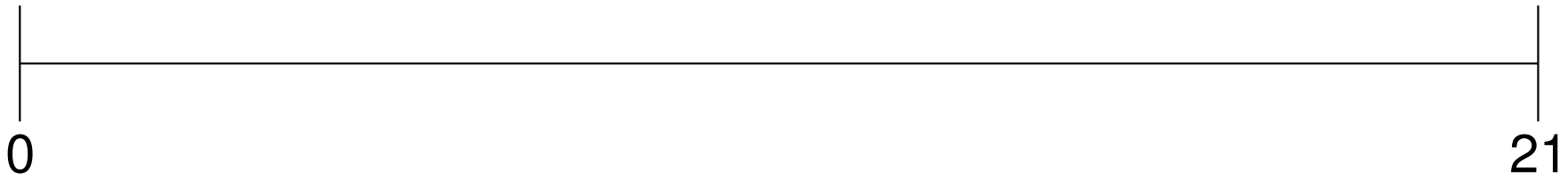
- select "characteristic time" $M > 0$ (should be $M = kh$, $k \in \mathbb{N}$) and $w \in \{2, 3, 4, \ldots\}$ (scaling parameter)
- for each primary mesh point $t_j$
  - find smallest integer $m$ such that $t_j < w^{m+1} M$
  - decompose
    $$[0, t_j] = [0, t_j - w^m M] \cup [t_j - w^m M, t_j - w^{m-1} M] \cup \cdots \cup [t_j - w^1 M, t_j - w^0 M] \cup [t_j - M, t_j]$$
  - define secondary mesh on each subinterval, starting from right:
    step size $h$ on first and second subintervals,
    step size $wh$ on third subinterval,
    step size $w^2 h$ on fourth subinterval, ...,
    step size $w^{m-1} h$ on penultimate subinterval,
    suitable combination of above step sizes on last subinterval

(Ford & Simpson 2001)

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$

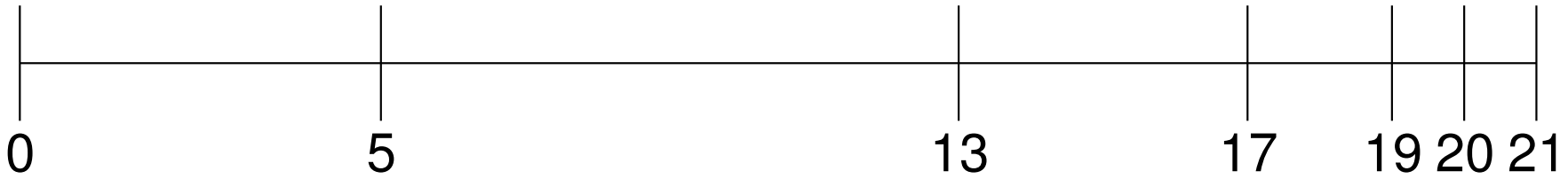



- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$




- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

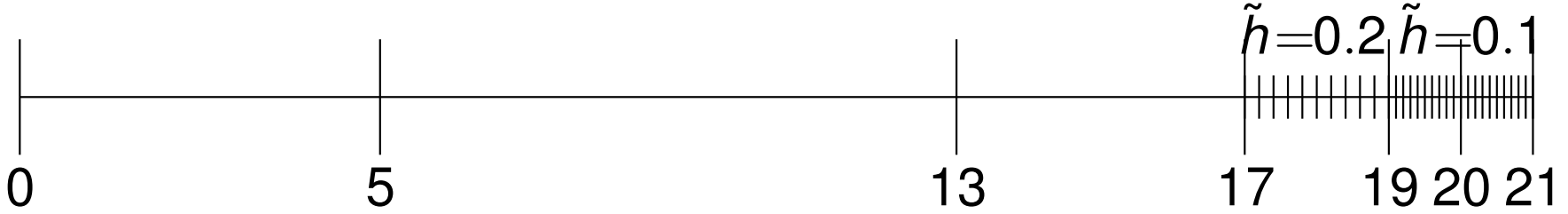
Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$


- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$

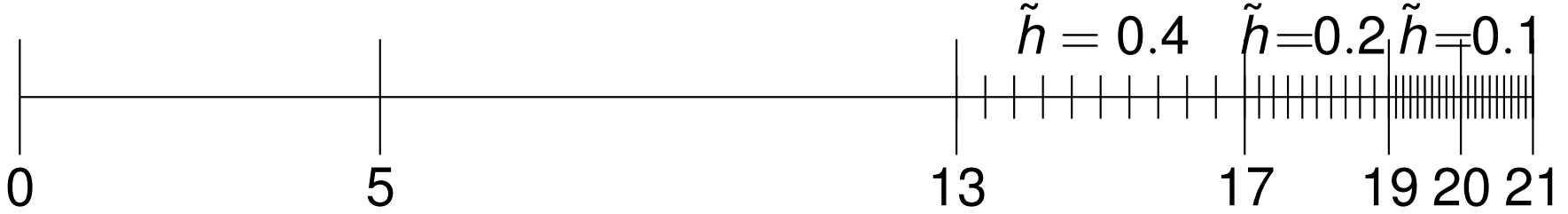



- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$

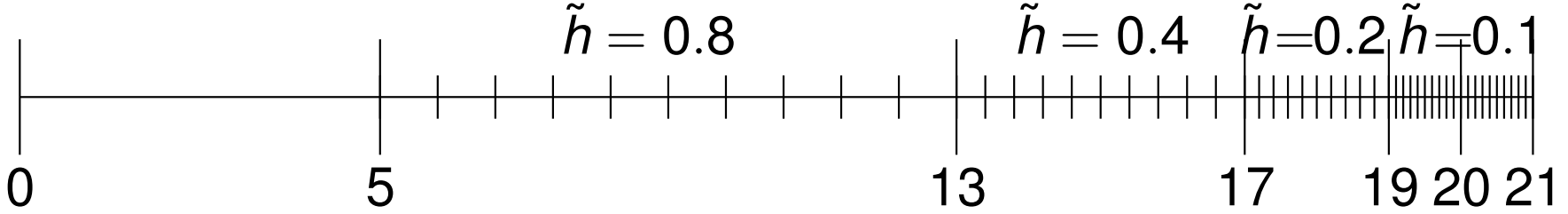



- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$

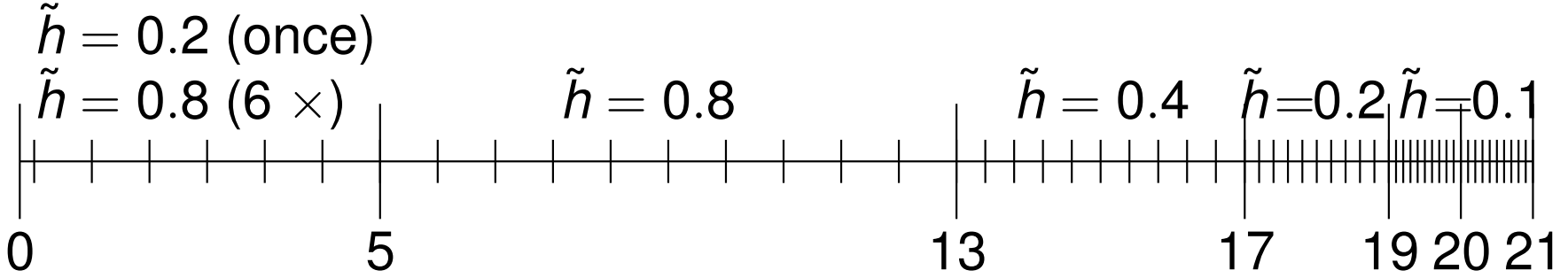



- Total number of nodes for uniform mesh: 211

# Logarithmic Memory I

Consequence: $N_j = O(\ln j)$

Example: $t_j = 21$, $h = 1/10$, $w = 2$ and $M = 1$




- Total number of nodes for uniform mesh: 211
- Total number of nodes for logarithmic mesh: 58

# Logarithmic Memory I

Overall error bound:

- Traditional approach (full secondary mesh):

$$ch^p \text{ with some } p > 0,$$

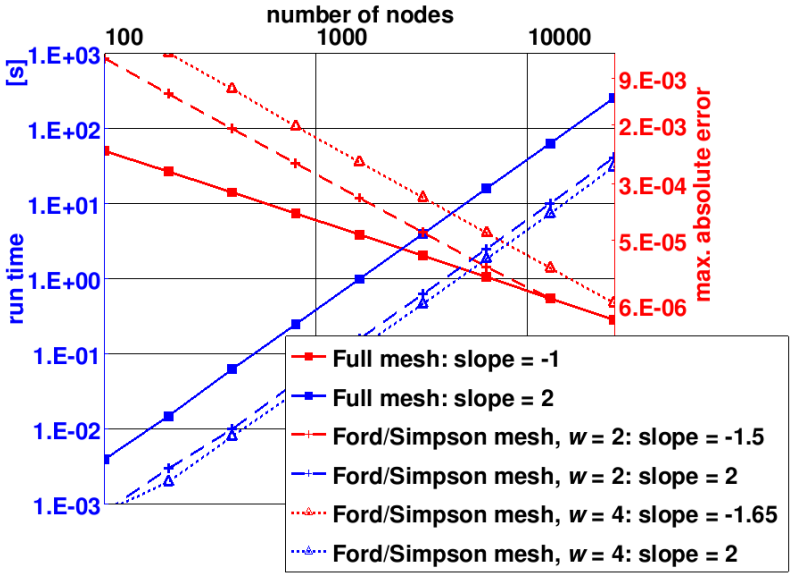
- Ford-Simpson logarithmic secondary mesh:

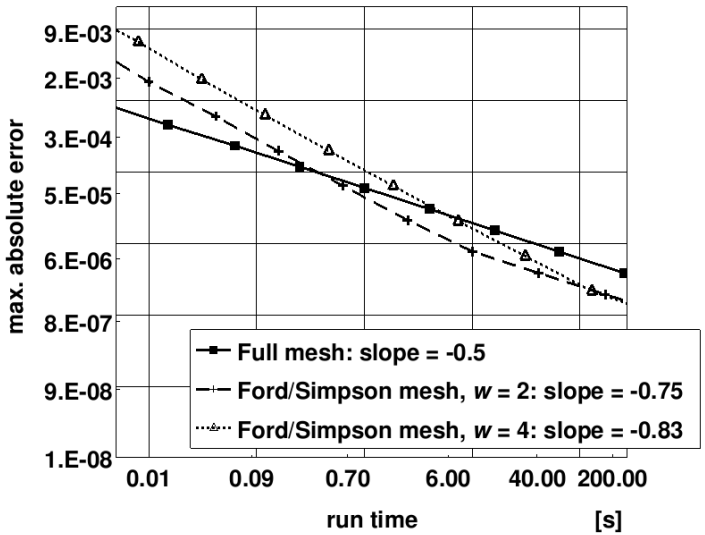
$$c\left(\frac{T}{Mw}h\right)^p = c'h^p \text{ with same } p$$

$\Rightarrow$ unchanged order of magnitude of error
but significantly reduced computational cost

# Logarithmic Memory I

Example: Smooth solution (case 1 above)




Similar results for other examples

# Logarithmic Memory II

Modification of Ford/Simpson idea (Di. & Freed 2006):
Given step size $h$ of primary mesh,

- select "characteristic time" $M = kh$ with fixed $k \in \{4, 5, \ldots\}$ and $w \in \{2, 3, 4, \ldots\}$ (scaling parameter)
- for each primary mesh point $t_j$
  - find smallest integer $m$ such that $t_j < w^{m+1} M$
  - decompose
    $$[0, t_j] \subset [t_j - w^{m+1}M, t_j - w^m M] \cup [t_j - w^m M, t_j - w^{m-1} M] \cup \cdots \cup [t_j - w^2 M, t_j - wM] \cup [t_j - wM, t_j]$$
    and set integrand $:= 0$ to the left of 0
  - define secondary mesh on each subinterval, starting from right:
    step size $h$ on first subinterval,
    step size $wh$ on second subinterval, ...,
    step size $w^{m-1}h$ on penultimate subinterval,
    step size $w^m h$ on modified last subinterval

# Logarithmic Memory II

Properties:

- $N_j = O(\ln j)$
- Very efficient memory management possible
- Closely related to panel clustering (McLean 2012)

# Shishkin Meshes

Equations with singular perturbations

Example:

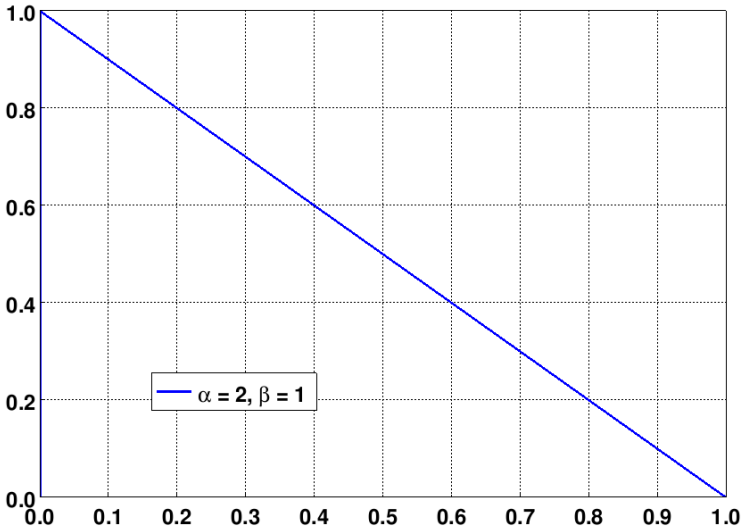
$$\begin{aligned} -\epsilon D^{\alpha}_{*0} y(t) - b(t) D^{\beta}_{*0} y(t) + c(t) y(t) &= f(t) \\ y(0) = y(1) &= 0 \end{aligned}$$

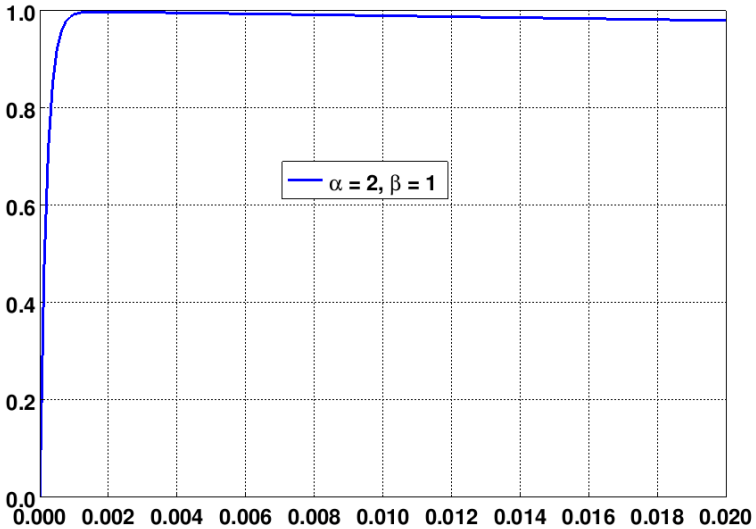
$$(0 < \beta \leq 1 < \alpha \leq 2; \quad \inf_{0 \leq t \leq 1} b(t) > 0, \quad c(t) \geq 0)$$

$\Rightarrow$ Solution exhibits boundary layer

# Shishkin Meshes

Classical case ($\beta = 1$, $\alpha = 2$):



# Shishkin Meshes

Classical case ($\beta = 1$, $\alpha = 2$):

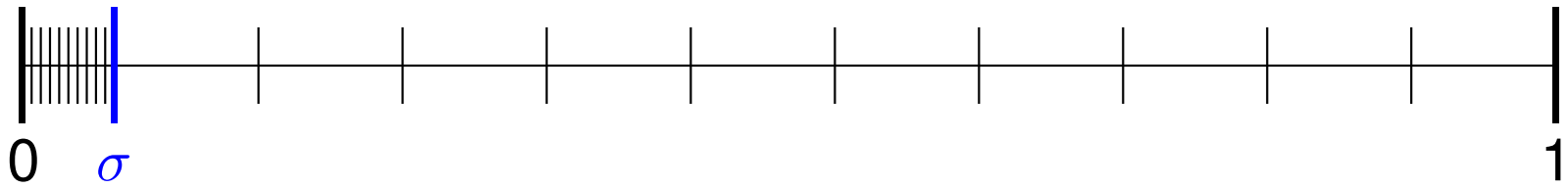
Use Shishkin mesh with $N$ subintervals ($\epsilon \ll N^{-1}$), i.e.

- introduce transition parameter $\sigma \in (0, 1)$
  (depending on $N$ and $\epsilon$; typically $\sigma = 2(\inf_t b(t))^{-1} \epsilon \ln N$)
- use $N/2$ equally large subintervals on $(0, \sigma)$,
- use $N/2$ equally large subintervals on $(\sigma, 1)$,
- apply (upwind) finite difference formula with this mesh.

(Roos, Stynes & Tobiska 2008; Linß 2010)

# Shishkin Meshes

Example: $\epsilon = 10^{-2}, N = 20, b(t) \equiv 1 \Rightarrow \sigma = 0.06$



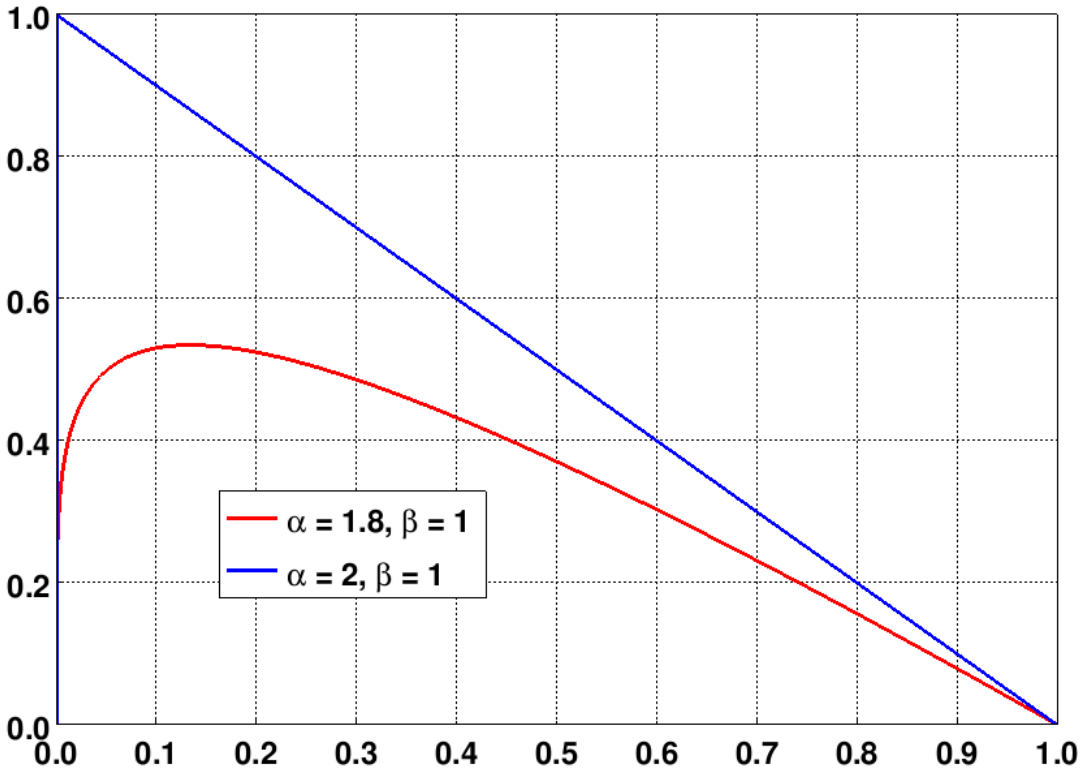

$\Rightarrow$ Fine (computationally expensive) mesh is used only where necessary.

# Shishkin Meshes

Example: Initial value problem

$$-10^{-4} D^{1.8}_{*0} y(t) - D^{1}_{*0} y(t) = -1, \quad y(0) = 0, y'(0) = 9199.08$$

Solution with uniform mesh:

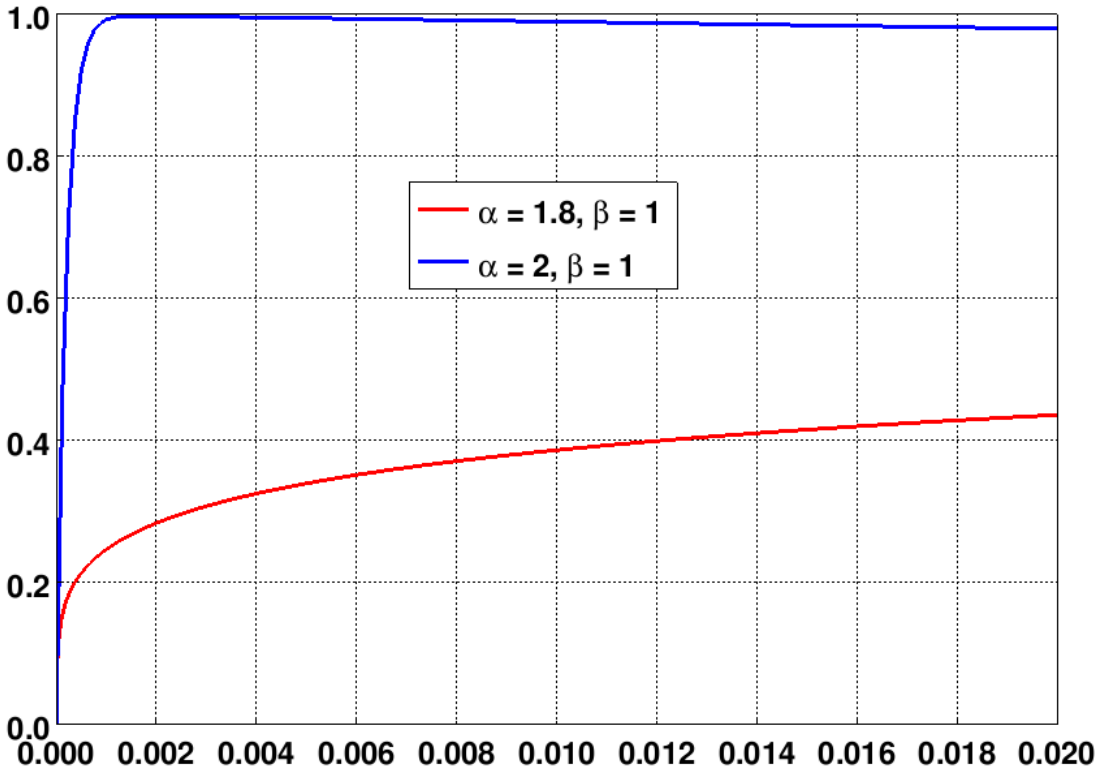


# Shishkin Meshes

Example: Initial value problem

$$-10^{-4} D^{1.8}_{*0} y(t) - D^{1}_{*0} y(t) = -1, \quad y(0) = 0, y'(0) = 9199.08$$

Solution with uniform mesh:

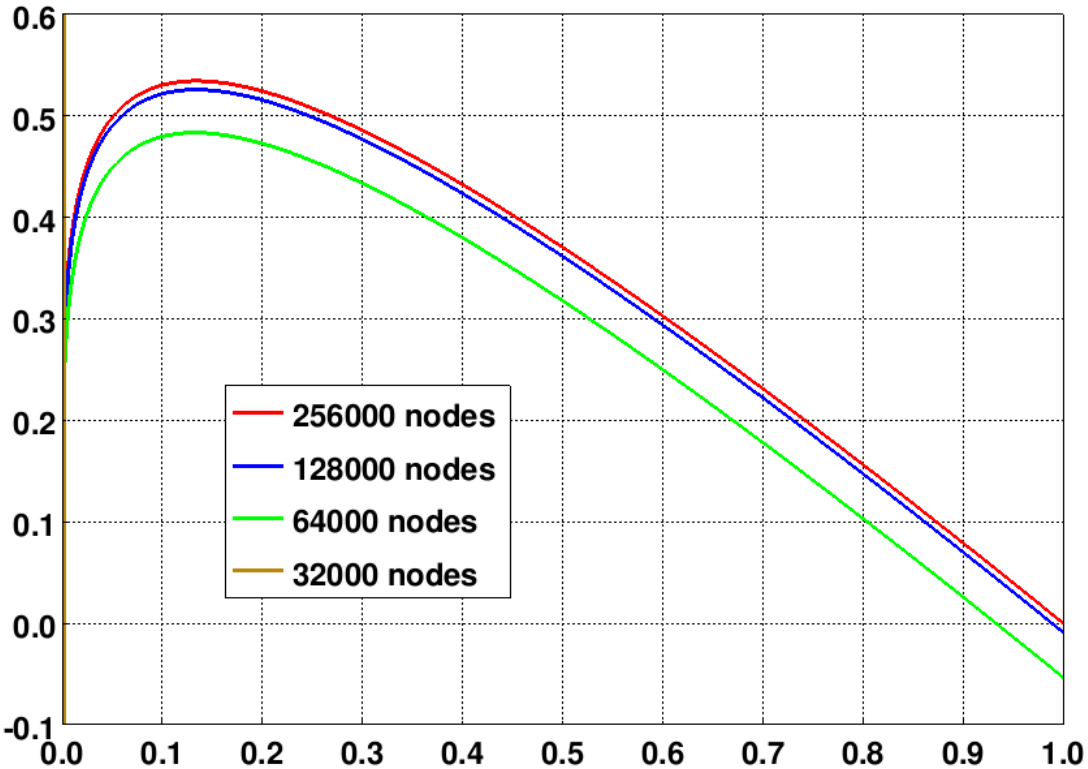


# Shishkin Meshes

Example: Initial value problem

$$-10^{-4} D_{*0}^{1.8} y(t) - D_{*0}^{1} y(t) = -1, \quad y(0) = 0, y'(0) = 9199.08$$

Solution with uniform mesh:



# Shishkin Meshes

Open questions in fractional case ($\alpha < 2$ or $\beta < 1$):

- Width of boundary layer
- Influence of changes of sign of coefficients (broken symmetry properties)
- Influence of memory on boundary layer (dependence of boundary layer on $\alpha, \beta$)
- Suitable choices for mesh (in particular: value of mesh transition parameter)

# Table of Contents

# Shared Memory Approach (OpenMP)

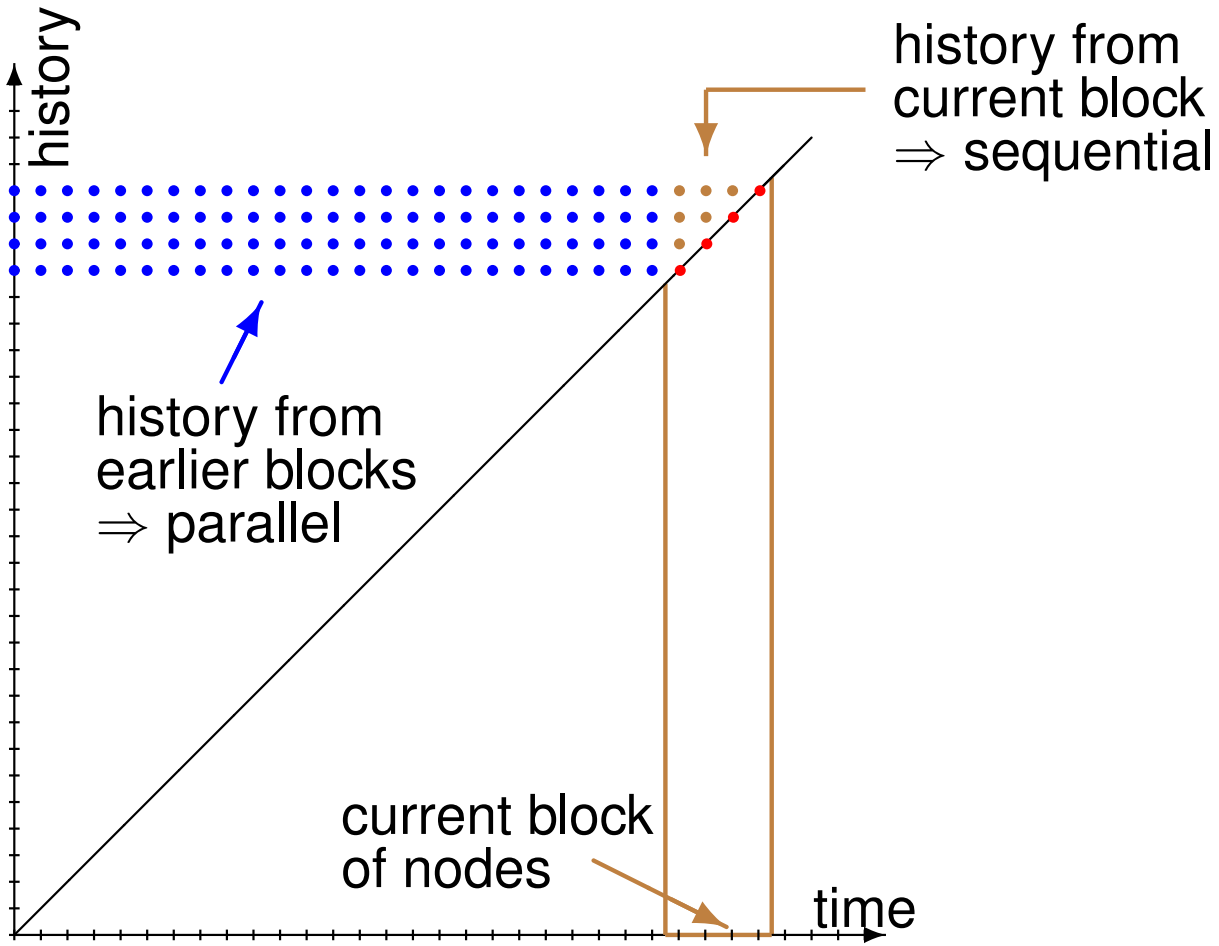
Basic idea for hardware platform with $P$ cores:

- Divide set of nodes into blocks of $P$ successive nodes each
- Handle blocks in parallel (one core per node of block):
  - Each node of block needs to take history into account (all previous nodes)
  - Split up history into nodes from earlier blocks and earlier nodes of current block
  - Compute first part of history in parallel for all nodes of current block
    (no interaction required; ideal scalability;
    main part of work except for the first few blocks)
  - Compute remainder of history sequentially
    (each node needs to wait for results of predecessors;
    small part of work only)

(Di. 2011)

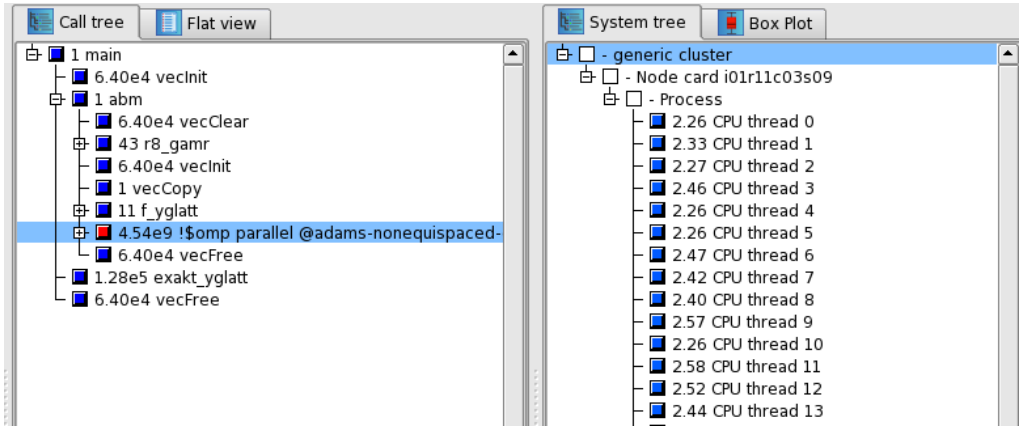
# Shared Memory Approach (OpenMP)

Graphical representation of strategy:



# Shared Memory Approach (OpenMP)

Performance analysis using Score-P measurement system (www.score-p.org)

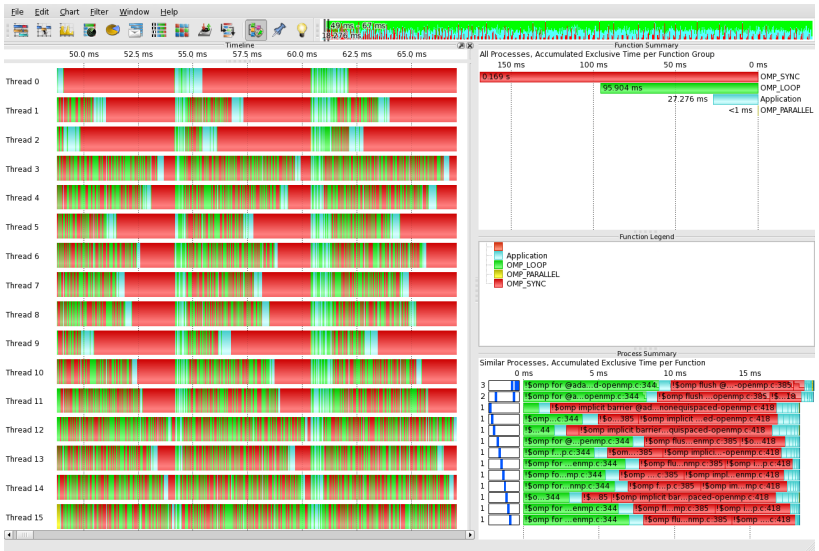


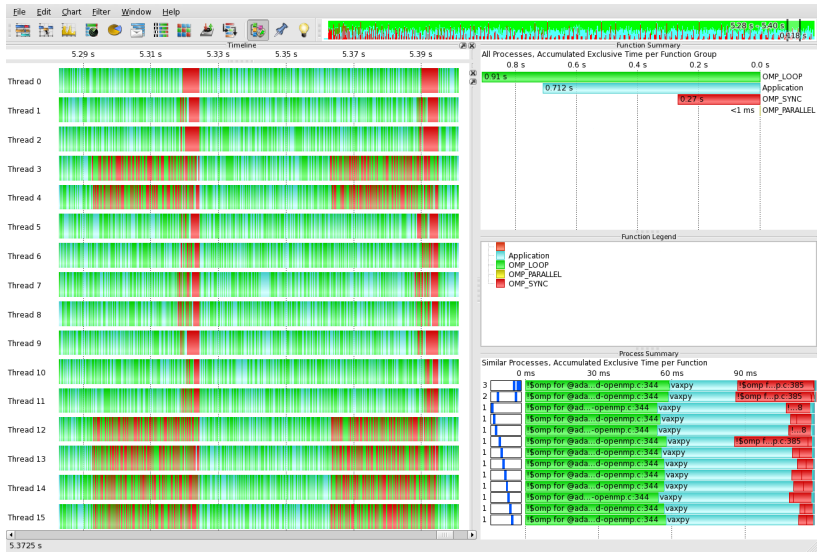

Profiling shows

- very good load balancing,
- very small sequential part

# Shared Memory Approach (OpenMP)

Performance analysis using Score-P measurement system (www.score-p.org)



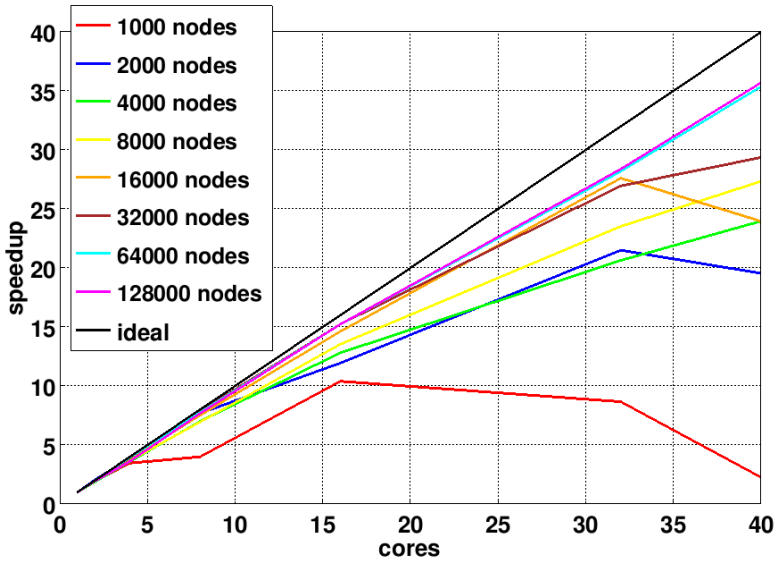



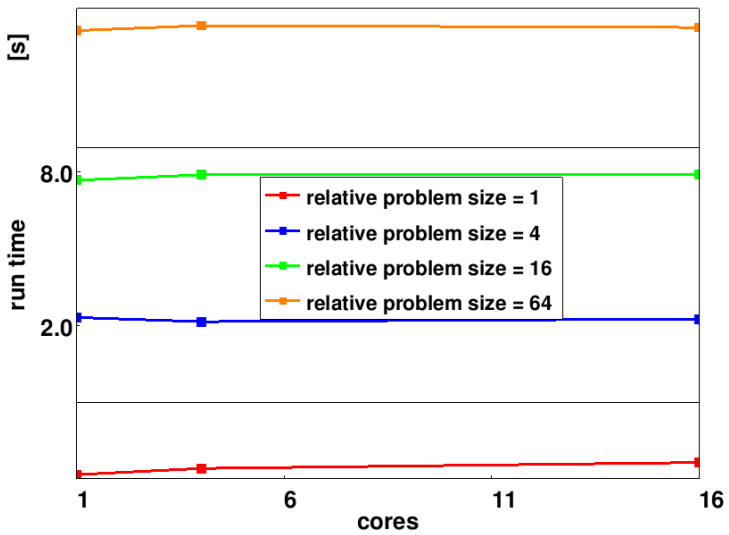

Tracing shows

- rather poor efficiency (much waiting time) only in very early part of simulation (left)
- very high efficiency (almost no waiting time) in later parts of simulation (right)

# Shared Memory Approach (OpenMP)

Observed performance:




- Very good strong (left) and weak (right) scaling
- Can be used for full memory or logarithmic memory
- Can be used for uniform or non-uniform primary meshes
- Same behaviour for corresponding approach on distributed memory systems (using MPI)



**Thank you for your attention!**

diethelm@gns-mbh.com

k.diethelm@tu-braunschweig.de