http://webcache.googleusercontent.com/search?...

The an Google search affigures an explication contribution that the contribution of th

1 of 3 04/10/2017 02:42 PM

Non-uniform first order finite differences (http://www.scientificpython.net/py	chlog/non-uniform-first-order-finite-differences). 3.Comments (http://www.scientific.got/non-aut/publica/plone-uniform-first-order-finite-differences/comments)	(http://www.statcount er.com)
Suppose we take a set of $n+1$ ordered, but non-uniformly distributed grid points $\{x_0, x_1, \dots, x_n\}$. If we have a function $f(x)$ the following terms of the following states as $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$ and $f(x) = f(x)$ are the following states as $f(x) = f(x)$.	that we can evaluate on these grid points. The function can be approximated as being piecewise parabolic. At a point x_k , approximate $f(x)$ with three quadratic Lagrange polynomials.	RSS Feed (/1/feed)
Let $h_k = x_k - x_{k-1}$. We can write the three polynomials as	$f(\mathbf{x}) \approx f(\mathbf{x}_{k-1})\ell'_{k-1}(\mathbf{x}) + f(\mathbf{x}_k)\ell'_{k}(\mathbf{x})f(\mathbf{x}_{k+1})\ell'_{k+1}(\mathbf{x})$	(http://www.linkedin.c om/in/veltri)
	$\ell_{k-1}(x) = \frac{x^2 - (x_{k+1} + x_k)x + x_k x_{k+1}}{h_k(h_k + h_{k+1})}$	Archives
	$\ell_k(x) = -\frac{x^2 - (x_{k+1} + x_{k-1})x + x_{k-1}x_{k+1}}{h_k h_{k+1}}$	February 2015 (/pyblog/archives /02-2015) January 2015 (/pyblog
	$\ell_{k+1}(x) = \frac{x^2 - (x_{k+1} + x_{k-1})x + x_k x_{k-1}}{(x_k + x_{k+1})x_{k+1}}$	Archives/01-20151
To numerically approximate the first derivative of f at the grid point x_k , we can differentiate the Lagrange interpolants		(/pyblog/archives /11-2014) September 2014 (/pyblog/archives /09-2014)
	$e''_{k-1}(x) = \frac{2\kappa - (x_{k+1} + x_k)}{h_k(h_k + h_{k+1})}$	
	$e'_{x}(x) = -\frac{2x - (x_{k+1} + x_{k-1})}{h_k h_{k+1}}$	/archives/08-2014) July 2014 (/pyblog /archives/07-2014) June 2014 (/pyblog /archives/06-2014)
	$\mathcal{E}_k(x) = -\frac{1}{h_k h_{k+1}}$	/archives/05-2014)
	$\mathscr{E}_{k+1}(x) = \frac{2x - (x_{k+1} + x_{k-1})}{(h_k + h_{k+1})h_{k+1}}$	April 2014 (/pyblog /archives/04-2014) March 2014 (/pyblog /archives/03-2014) February 2014
For all interior grid points, we approximate $f'(x_k)$ as	Com . additionals	February 2014 (/pyblog/archives /02-2014)
	$f'(x_k) \approx a_k f(x_{k-1}) + b_k f(x_{k-1}) + c_k f(x_{k+1})$	January 2014 (/pyblog /archives/01-2014)
where the weights are obtained by evaluating the derivatives of the Lagrange polynomials at x_k	$a_k = \ell'_{k-1}(u_k) = -\frac{h_{k+1}}{h_k(b_k + h_{k+1})}$	December 2013 (/pyblog/archives /12-2013) November 2013 (/pyblog/archives
	$b_1 = a'_{1k}(a_k + b_{2k+1})$ $b_k = a''_{1k}(a_k) = \frac{b_{1k+1} - b_{1k}}{b_{1k}b_{2k+1}}$	/11-2013) October 2013
	$c_k = c'_{k+1}(n_k) = \frac{h_k}{h_{k+1}}$ $c_k = c'_{k+1}(n_k) = \frac{h_k}{h_{k+1}(h_k + h_{k+1})}$	/10-2013) September 2013 (/pyblog/archives (09-2013) August 2013 (/pyblog
There formulae held for hor 1 and 1	$h_{k+1}(h_k + h_{k+1})$	
These formulas hold for $k=1,\ldots,n-1$ We have to do something different for $k=0$ and $k=n$, so we use one-sided differences	$f'(s_0) \approx a_0f(s_0) + b_0f(s_1) + c_0f(s_2)$	July 2013 (/pyblog /archives/07-2013) June 2013 (/pyblog /archives/06-2013) April 2013 (/pyblog
with	י עובי אויי י	April 2013 (/pyblog /archives/04-2013) March 2013 (/pyblog
	$a_0 = -rac{2h_1 + h_2}{h_1(h_1 + h_2)}$	/archives/03-2013) February 2013
		(/pyblog/archives /02-2013) January 2013 (/pyblog /archives/01-2013) December 2012
	$h_0=\frac{h_1+h_2}{h_1h_2}$	(/pyblog/archives
	$c_0 = -\frac{h_1}{h_2(h_1 + h_2)}$	(12-2012) November 2012 (fpyblog/archives (11-2012) September 2012 (fpyblog/archives (09-2012) August 2012 (fpyblog/archives) (pyblog/archives)
Similarly at the right end, we have	$n_2(n_1+n_2)$	(/pyblog/archives /09-2012) August 2012 (/pyblog /archives/08-2012)
	$a_a = \frac{h_a}{h_{a-1}(h_{a-1} + h_a)} b_a = -\frac{h_{a-1} + h_a}{h_{a-1}h_a} c_a = \frac{2h_a + h_{a-1}}{h_a(h_{a-1} + h_a)}$	July 2012 (/pyblog /archives/07-2012) June 2012 (/pyblog
To numerically differentiate f on the grid, we can apply an almost tridiagonal matrix ${\bf D}$ to the sampled function.		July 2012 (Jpyhlog /archives/07-2012) lune 2012 (Jpyhlog /archives/06-2012) May 2012 (Jpyhlog /archives/05-2012)
Where the differentiation matrix is	$\mathbf{f}' = \mathbf{D}\mathbf{f}$	/archives/04-2012)
where the differentiation matrix is	$\begin{pmatrix} a_0 & b_0 & c_0 \end{pmatrix}$	Categories All (/pyblog
	$\mathbf{D} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix}$	All (PD/DIOG) (category/all) Basis Functions (/pyblog/category /basis functions) Biharmonic (/pyblog
	$a_{n-1} \ b_{n-1} \ c_{n-1}$	(hibanania)
Below is a Python code which, when given the vector \mathbf{x} , produces the matrix \mathbf{D} as a sparse matrix.	$\begin{bmatrix} a_n & b_n & c_n \end{bmatrix}$	Boundary Conditions (/pyblog/category /boundary conditions) Boundary Value Problem (/pyblog
import numpy as np from scipy.sparse import csr_matrix		Boundary Value Problem (/pyblog /category/boundary value problem)
tron scipty-spane import or matrix def mid(x) n = len(x) = (1 + (1 + (1 + (1 + (1 + (1 + (1 + (1		Boundary Value Problems (/pyblog /category/boundary value problems)
$a0 = (2^nh(0)+h[1])(h[0]^n(h[0]+h[1]))$ $ak = h[1]/(h[-2]^n(h[-2]+h[1]))$ $an = h[-1]/(h[-2]^n(h[-1]+h[-2]))$		(category/chebyshev)
b0 = (a(1)+b(1),a(a(1)+a(1)) bk = (b(1)-b(1)-b(1)+b(1)) bn = (b(1)+b(2))(b(1)+b(1)) c0 = b(0)(b(1)+b(0)(1)+b(1))		Differentiation (/pyblog/category /differentiation)
$ck = h[:n\cdot 2]/(h[:1:]^n[h::n\cdot 2]+h[:1])$ $cn = (2^nh[:1]+h[:2])/(h[:1]^n[h::])$ val = np. hatack((n) d.k.n.b, 0 b.k.h.n.c) c.k.,cn))		(differentiation) Discontinuous Galerkin (/pyblog /category /discontinuous
row = pp.tile(np.arange(n),3) dex = np.hstack((0p.arange(n/2),n·3)) col = np.hstack((dex,dex+1,dex+2))		galerkin) Domain
D = 0.00 interfaction(con), snape=(0.0)) return D (uploads/1/1/5/9/11598566/nufd.py)	mold py Download File (uploads/1/1/5/9/11598566/mufd.py)-dex	Decomposition (/pyblog/category (domain decomposition)
Total	pownload THE (*uploads*) 11/29/21/12983-90/.nuta.pypess	Eigenvalue Problems (/pyblog/category /eigenvalue problems)
3 Comments (http://www.scientificpython.net/python/non-uniform-first-order-finite-differences#comments)		Fast Transform (/pyblog/category/fast transform) Finite Difference
caduqued	6/10/21/2 Ct 16:53 sm	/finite difference)
Really rice alter and the into is extremely clear! Thanks for sharing!	Daply	(/pyblog/category /finite differences) Finite Element (/pyblog/category
joss	1202012 0x 1951 am	/hnite element) Fourth Order (/pyblog
Should cop_matrix be car_matrix? Thenks for the irreferentation though! Have bookmarked your alts - It looks extremely interesting!	Bayly	/category/fourth order)
Ben	3500010 12.01 63 am	(category/galerkin) Galerkin Method (/pyblog/category /galerkin method)
Wary helpful filterios: a armill correction to the expression for the first derivative: f(six-skipts-1) = bit(pls-1) = clt(pls-1) and pls- second by the second pls-second pls-		(galerkin method) Initial Value Problems (/pyblog/category /initial value
should be $(1/a)$ sub $(1/a)$ -		problems) Integral Equation (/pyblog/category
	порід	(integral equation) Integration (/pyblog (category/integration) Iterative Solver
Your comment will be posted after it is approved.		(/pyblog/category /iterative solver) Jacobi Polynomials (/pyblog/category
Leave a Reply.		(/pyblog/category /jacobi polynomials) Kronecker Product (/pyblog/category
Nama (required)		/krylov Method (/pyblog/category /krylov method)
Email (not published)		Interpolation (/pyblog
Website		/category/lagrange interpolation) Lagrange Polynomial (/pyblog/category
Comments		(lagrange polynomial) Legendre (/pyblog
		(category/legendre) Manufactured Solution (/pyblog /category /manufactured
Notify me of new comments to this post by email Subsolt		Modal (/pyblog
		Newton (/pyblog /category/newton) Nodal Galerkin
		(/pyblog/category /nodal galerkin) Nonuniform (/pyblog
		/category /nonuniform) Numerical Differentiation
		/pyblog/category /numerical differentiation) Numerical
		/category/numerical
		integration) Optimization (/pyblog /category

2 of 3 04/10/2017 02:42 PM

Flexima (Iraphon extraordina) in the control of the

Copyright 2013 Greg von Winckel All Rights Reserved

3 of 3 04/10/2017 02:42 PM