

Numerical Differentiation

motivation

- ▶ sections on numerical differentiation and integration mirror core topics of calculus
- ▶ approximations of derivatives used in
 - ▶ solution of nonlinear equations
 - ▶ solution of ordinary and partial differential equations
 - ▶ data analysis
- ▶ determination of derivatives from measured data is archetypical example of an *ill-posed problem*
- ▶ problem: determine $f'(x)$ from finite number of values $f(x_i)$ which may contain errors
- ▶ use definition from calculus as a guide:

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}.$$

- ▶ approach: find method which is exact for polynomials

truncation error

- Taylor expansion

$$\begin{aligned}\frac{f(x+h) - f(x)}{h} &= \frac{[f(x) + hf'(x) + h^2 f''(\xi)/2] - f(x)}{h} \\ &= f'(x) + \frac{f''(\xi)}{2} h\end{aligned}$$

for some $\xi \in [x, x+h]$

- for bounded f'' and $h \rightarrow 0$

$$e_h(x) := \frac{f(x+h) - f(x)}{h} - f'(x) = O(h) \rightarrow 0$$

this is the *truncation error*

- question for you: what is the relative truncation error?
- also: consider rounding error!

example $f(x) = x \sin(x)$ for $x = 1$

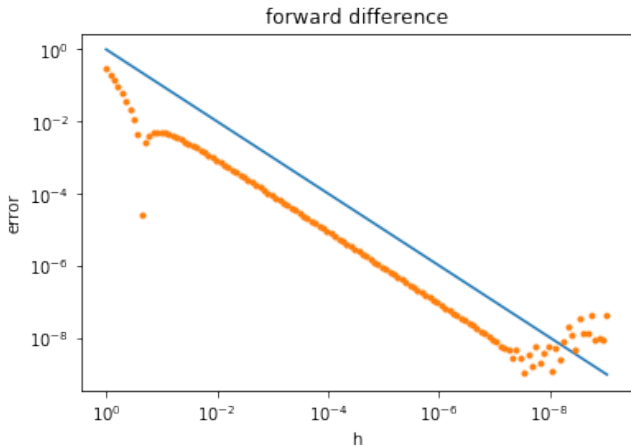
- ▶ exact derivative $f'(x) = \sin(x) + x \cos(x)$
- ▶ rel. error

$$e_h = \left| \frac{\frac{f(x+h)-f(x)}{h} - f'(x)}{f'(x)} \right|$$

```
f = lambda x : x*np.sin(x)
df = lambda x : np.sin(x) + x*np.cos(x)
D = lambda f,x,h : (f(x+h) - f(x))/h

h = 2**(-np.linspace(0,30,128))
```

```
plt.title('forward difference')
plt.gca().invert_xaxis();plt.loglog(h,h)
plt.xlabel('h');plt.ylabel('error')
plt.loglog(h,abs((D(f,1.0,h) - df(1.0))/df(1.0)),'.');
```



finite difference approximations

first order approximations of first derivative

Consider the two Taylor series expansions

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4),$$

and

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4).$$

- ▶ forward difference approximation

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

- ▶ backward difference approximation

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h)$$

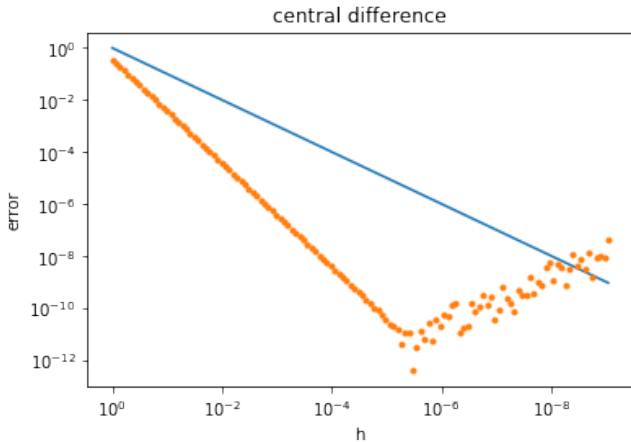
second order approximations of first derivative

- *central difference approximation*

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - O(h^2)$$

Dc = `lambda f,x,h : (f(x+h) - f(x-h))/(2*h)`


```
plt.title("central difference")
plt.gca().invert_xaxis();plt.loglog(h,h);plt.xlabel('h')
plt.ylabel('error');plt.loglog(h,abs((Dc(f,1.0,h) - df(1.0)
```



second derivatives

To find an approximation of the second derivatives further expansion of the Taylor series is required,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f''''(x) + O(h^5),$$

and

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f''''(x) + O(h^5)$$

Add the equations to get

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{12}f''''(x) + O(h^5)$$

So

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2 f''''(x)}{12} + O(h^3)$$

The Method of Undetermined Coefficients

method of undetermined coefficients

- ▶ aim: compute approximation $D_h(f, x) \approx f'(x)$ using function values $f(x_k)$ at *grid points* x_k by *finite difference approximation*

$$D_h(f, x) = \sum_{k=0}^n c_k f(x_k)$$

- ▶ approach: compute coefficients such that formula is exact for polynomials p of degree up to n

$$D_h(p, x) = p'(x)$$

- ▶ equidistant grid: consider $x_k = x_0 + kh$
- ▶ transformation $z = x_0 + xh$, $g(x) = f(x_0 + xh)$ and

$$g'(x) = hf'(x_0 + xh)$$

- ▶ thus determine coefficients c_j for case $x_k = k$ only and use transformation
- ▶ special case $x = n/2$: central difference

determination of coefficients c_j for case $x_k = k$ and $n = 2$

- ▶ consider monomials $p(x) = 1, x, x^2, \dots$
- ▶ system of equations with Vandermonde matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2x \end{bmatrix}$$

- ▶ Gaussian elimination

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2x - 1 \end{bmatrix}$$

- ▶ solution

$$c_2 = x - 1/2, \quad c_1 = 2(1 - x), \quad c_0 = x - 3/2$$

computing coefficients for central differences

- ▶ solve linear system of equations $Vc = b$ where V is Vandermonde matrix with elements k^j
- ▶ central differences: $x = n/2$
- ▶ Vandermonde matrix using numpy's vander function
- ▶ rhs b with components $b_k = k(n/2)^{k-1}$ for $k = 0, 1, \dots, n$

```
n = 3
V = np.vander(np.arange(n+1), increasing=True).T
b = np.arange(n+1)*(n/2)**(np.arange(n+1)-1)
ck = np.linalg.solve(V,b)
print(ck)

[ 0.04166667 -1.125          1.125         -0.04166667]
```

example $f'(1)$ for $f(x) = x \sin(x)$ using central difference

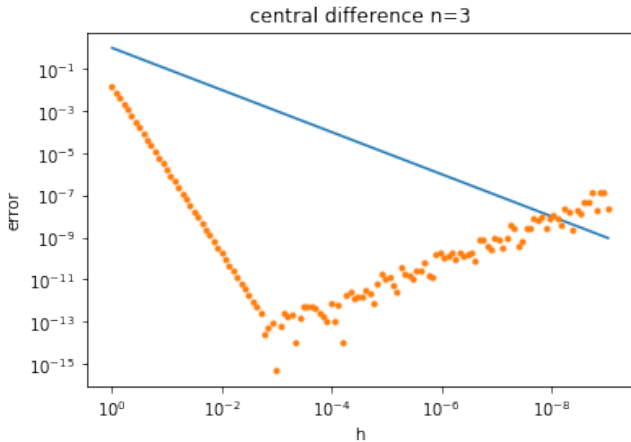
- ▶ (transformed) grid points for central difference

$$x_k = 1 + (k - n/2)h$$

- ▶ transform coefficients $c_k \rightarrow c_k/h$

```
def Dn(f, x, h, ck=ck):  
    n = ck.shape[0] - 1  
    ydot = 0  
    for k in range(n+1):  
        ydot += ck[k]/h*f(x+(k-n/2)*h)  
    return ydot
```

```
plt.title("central difference n={}".format(n))  
plt.gca().invert_xaxis();plt.loglog(h,h)  
plt.xlabel('h');plt.ylabel('error')  
plt.loglog(h,abs((Dn(f,1.0,h) - df(1.0))/df(1.0)),'.');
```



Richardson Extrapolation

extrapolating central difference

- ▶ central difference $\phi(h) = (f(x+h) - f(x-h))/(2h)$
 - ▶ $\phi(h)$ is even function, Taylor series contains only terms with h^{2j}
 - ▶ if f continuously differentiable, then $\phi(h)$ can be continuously continued so that $\phi(0) = f'(x)$
- ▶ interpolating polynomial of $\phi(kh)$ for $k = 1, 2$ as function of k^2 (Lagrange formula!)

$$p(k) = \frac{(4 - k^2)\phi(h) + (k^2 - 1)\phi(2h)}{4 - 1}$$

- ▶ *extrapolation*: evaluate this polynomial for $k = 0$

$$p(0) = \frac{4\phi(h) - \phi(2h)}{3}$$

extrapolation and elimination

- ▶ Taylor expansions of $f(x \pm h)$ give

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(x)h^2}{6} + O(h^4)$$

- ▶ evaluate this for both h and $2h$

$$f'(x) = \phi(h) + a_2h^2 + a_4h^4 + O(h^6)$$

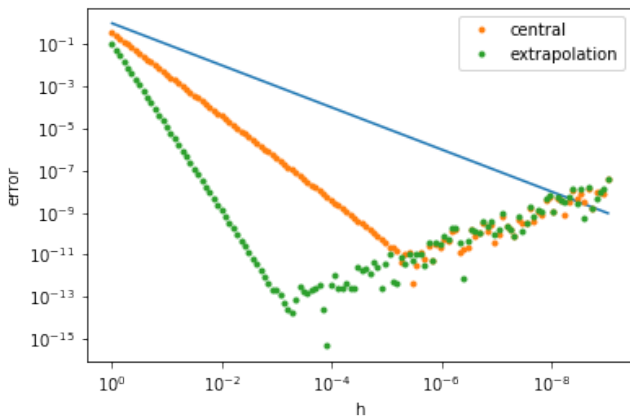
$$f'(x) = \phi(2h) + 4a_2h^2 + 16a_4h^4 + O(h^6)$$

- ▶ unknowns: $f'(x), a_2, a_4, \dots$
- ▶ idea: eliminate a_2

$$f'(x) = \frac{1}{3}[4\phi(h) - \phi(2h) - 12a_4h^4] + O(h^6)$$

error of extrapolation

```
R = lambda f,x,h: (4*Dc(f,x,h) - Dc(f,x,2*h))/3  
plt.gca().invert_xaxis();plt.loglog(h,h);plt.xlabel('h');p  
plt.loglog(h,abs((Dc(f,1.0,h) - df(1.0))/df(1.0)),'. ',label=  
plt.loglog(h,abs((R(f,1.0,h) - df(1.0))/df(1.0)),'. ',label=
```



Richardson's idea

- ▶ sequentially eliminate $a_2, a_4, a_6 \dots$ by including values $\phi(4h), \phi(8h), \dots$
- ▶ after eliminating a_2 we get

$$f'(x) = \frac{1}{3}[4\phi(h) - \phi(2h) - 12a_4h^4] + O(h^6)$$

- ▶ introduce matrix R with $R(n, 0) = \phi(2^n h)$ for $n = 0, 1, 2, \dots$
- ▶ furthermore $R(n, 1) = (4R(n, 0) - R(n+1, 0))/3$ so that

$$f'(x) = R(0, 1) - 4a_4h^4 + O(h^6)$$

and

$$f'(x) = R(1, 1) - 4^2 * 4a_4h^4 + O(h^6)$$

- ▶ now eliminate a_4 to get

$$f'(x) = (4^2 R(0, 1) - R(1, 1))/15 + O(h^6)$$

- ▶ thus set $R(n, 2) = (4^2 R(0, 1) - R(1, 1))/15$

the general scheme

- ▶ start with

$$R(n, 0) = \phi(2^n h)$$

- ▶ then set

$$R(n, 1) = \frac{4R(n, 0) - R(n+1, 0)}{3}$$

- ▶ and repeat to get

$$R(n, k) = \frac{4^k R(n, k-1) - R(n+1, k-1)}{4^k - 1}$$

until desired accuracy achieved

- ▶ show that this eliminates a_2, a_4, a_6, \dots using Taylor expansion
- ▶ this can also be applied to other functions $\phi(h)$

the tableau

- ▶ all the $R(i, j)$ required for $R(0, k)$:

$$\begin{array}{ccccccc} R(0, 0) & R(0, 1) & R(0, 2) & \cdots & \cdots & & R(0, k) \\ R(1, 0) & R(1, 1) & R(1, 2) & \cdots & R(1, k-1) & & \\ \vdots & \vdots & \vdots & \ddots & & & \\ R(k-2, 0) & R(k-2, 1) & R(k-2, 2) & & & & \\ R(k-1, 0) & R(k-1, 1) & & & & & \\ R(k, 0) & & & & & & \end{array}$$

- ▶ follows from

$$R(n, k) = \frac{4^k R(n, k-1) - R(n+1, k-1)}{4^k - 1}$$

- ▶ use $R(n, k)$ for error estimate

$$e_k = R(0, k) - \phi(0) \approx R(0, k) - R(0, k+1)$$

example

```
mr = 5; h = 0.1
R = np.zeros((mr,mr))
for n in range(mr):
    R[n,0] = Dc(f,1.0,2**n*h)
for k in range(1,mr):
    for n in range(mr-k):
        R[n,k] = (4**k*R[n,k-1] - R[n+1,k-1])/(4**k-1)
print(R)
print("exact value: {}".format(df(1.0)))

[[1.37666939 1.38175749 1.38177321 1.38177329 1.38177329]
 [1.36140508 1.38152171 1.38176814 1.38177306 0.          ]
 [1.30105517 1.37782526 1.38145793 0.          0.          ]
 [1.07074492 1.32333509 0.          0.          0.          ]
 [0.3129744  0.          0.          0.          0.          ]]
exact value: 1.3817732906760363
```


errors

```
for n in range(mr):  
    for k in range(mr-n):  
        print("{:1.2e}".format(R[n,k]-df(1.0)),end='    ')  
    print("\n")
```

-5.10e-03 -1.58e-05 -8.14e-08 -9.07e-10 -2.51e-11

-2.04e-02 -2.52e-04 -5.15e-06 -2.26e-07

-8.07e-02 -3.95e-03 -3.15e-04

-3.11e-01 -5.84e-02

-1.07e+00

error approximation for first row in R

```
# error approximation for first row
print("error approx",end=' ')
for k in range(mr-1):
    print("{:1.2e}".format(R[0,k]-R[0,k+1]),end=' ')
print("\n error exact",end=' ')
for k in range(mr-1):
    print("{:1.2e}".format(R[0,k]-df(1.0)),end=' ')
```

error approx	-5.09e-03	-1.57e-05	-8.05e-08	-8.82e-10
error exact	-5.10e-03	-1.58e-05	-8.14e-08	-9.07e-10