

Math 128a, Section 3 — Solutions to Problem Set 1

(1) Problem 1.5.1

The hard part has already been done for us; we are told that we can model the error by

$$\overline{errC}(h) = \frac{M_3 h^2}{6} + 2\frac{\delta}{h}.$$

Using a bit of calculus, we find the optimum value h_{optC} ($\neq h_{opt}$) to use in computing the central divided difference.

$$\frac{d}{dh} \left(\frac{M_3 h^2}{6} + 2\frac{\delta}{h} \right) = \frac{M_3 h}{3} - 2\frac{\delta}{h^2} = 0$$

gives

$$h_{optC} = \sqrt[3]{\frac{6\delta}{M_3}}, \quad \overline{errC}(h_{optC}) = 3\sqrt[3]{\frac{M_3\delta^2}{6}}.$$

We should use the new (central) style of divided difference rather than the old whenever

$$\overline{errC}(h_{optC}) < \overline{errD}(h_{opt})$$

is satisfied; this is equivalent to

$$81\delta(M_3)^2 < 256(M_2)^3.$$

```

http://www.math.berkeley.edu/~hthall/128A/matlab/Derivative2.m

function [d,err] = Derivative2(fname,a,delta,M2,M3)
% d = Derivative(fname,a,delta,M2,M3);
% fname a string that names a function f(x) whose derivative
% at x = a is sought. delta is the absolute error associated with
% an f-evaluation and M2 is an estimate of the second derivative
% magnitude near a, and M3 is an estimate of the third derivative.
% d is an approximation to f'(a) using the better of forward or
% central divided difference, and err is an estimate
% of its absolute error.
%
% Usage:
%   [d,err] = Derivative(fname,a)
%   [d,err] = Derivative(fname,a,delta)
%   [d,err] = Derivative(fname,a,delta,M2)
%   [d,err] = Derivative(fname,a,delta,M2,M3)
%
% Caveat: The original function Derivative didn't do any
% error checking (e.g. delta, M2, and M3 should be positive),
% so neither are we.

if nargin <= 4
    % for no good reason, guess that

```

```

    % third derivative bound is 1.
    M3 = 1;
end
if nargin <= 3
    % No derivative bound supplied, so assume the
    % second derivative bound is 1.
    M2 = 1;
end
if nargin == 2
    % No function evaluation error supplied, so
    % set delta to eps.
    delta = eps;
end
% Which method is better?
if 81*delta*M3^2 < 256*M2^3

% Compute optimum h and central divided difference
    hopt = (6*delta/M3)^(1/3);
    d = (feval(fname,a+hopt) - feval(fname,a-hopt))/2/hopt;
    err = 3*(M3*delta^2/6)^(1/3);

else

% Compute optimum h and forward divided difference
    hopt = 2*sqrt(delta/M2);
    d = (feval(fname,a+hopt) - feval(fname,a))/hopt;
    err = 2*sqrt(delta*M2);

end

```

(2) Problem 1.6.1

<http://www.math.berkeley.edu/~hthall/128A/matlab/Dot3.m>

```

function s = Dot3(x,y)
% x = Dot3(x,y)
% x and y are column vectors of the same length. s is their
% dot product, computed as though MATLAB's internal Representation
% was three-digit base-10 floating point.

% accumulator
acc = Represent(0);

% just a little error checking
len = min(length(x), length(y));

```

```

for k=1:len
    acc = Float(acc,
                Float(Represent(x(k)), Represent(y(k)), '*'), ...
                '+'');
end

s = Convert(acc);

```

<http://www.math.berkeley.edu/~hthall/128A/matlab/Dot3Hist.m>

```

% Script File: Dot3Hist
% Histogram of 3-digit floating point error
% in computing 5-tuple dot products.

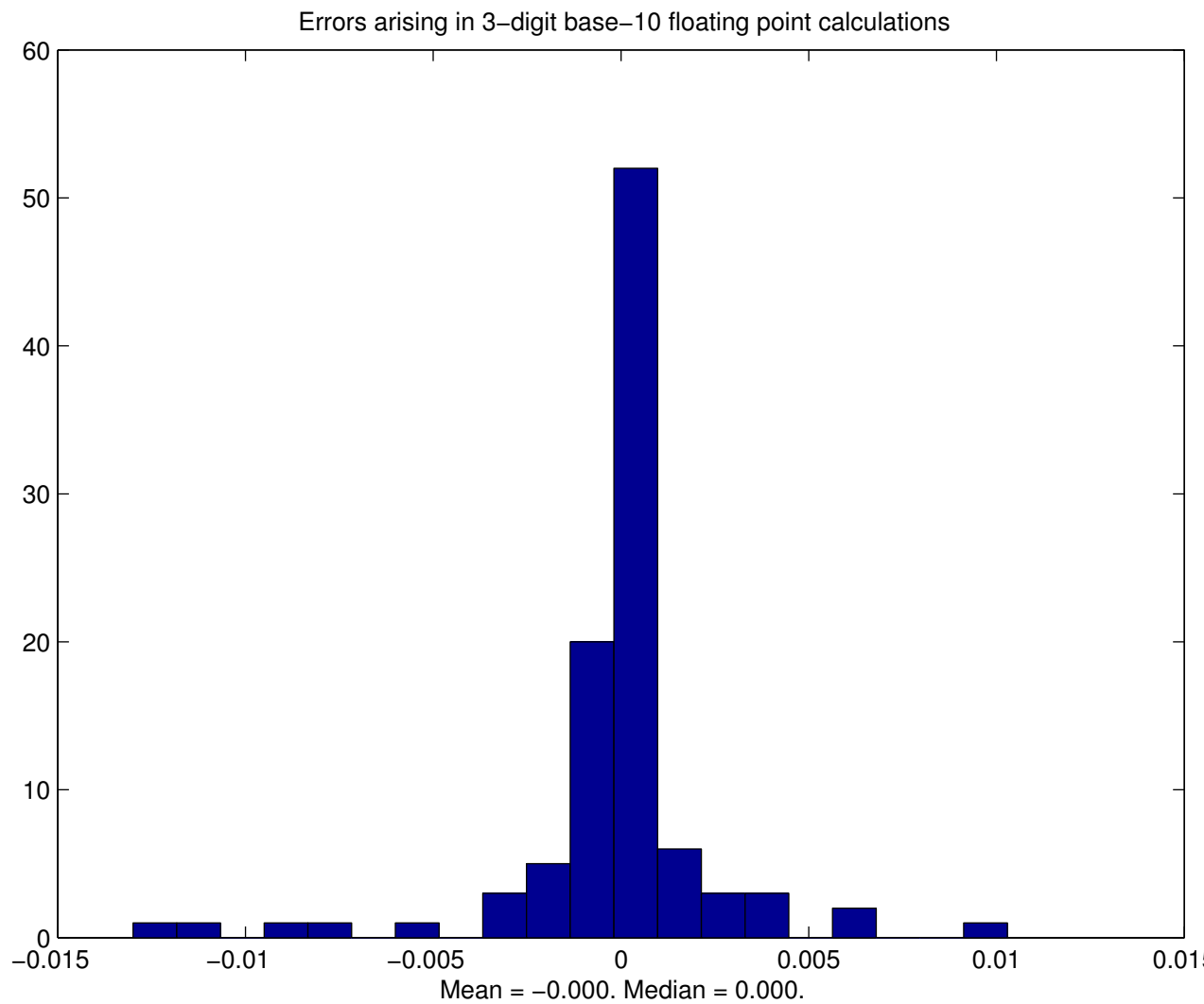
close all
x = randn(5,100);
y = randn(5,100);
er = zeros(100,1);

for k = 1:100
    er(k) = x(k)'*y(k)-Dot3(x(k),y(k));
end

hist(er,20)
% axis([-1 1 0 100])
title('Errors arising in 3-digit base-10 floating point calculations')
xlabel(sprintf('Mean = %5.3f. Median = %5.3f.',mean(er),median(er)))

```

Here's a typical result:



(3) Problem 1.7.1

<http://www.math.berkeley.edu/~hthall/128A/matlab/Arch.m>

```
function arch(a,b,theta1,theta2,r1,r2,ring_color)
```

```
% Adds an arch with center (a,b), inner radius r1, and outer radius r2 to the current figure.
% The arch is the set of all points of the form (a+r*cos(theta),b+r*sin(theta)) where
% r1 <= r <= r2 and theta1 <= theta <= theta2, with theta1 and theta2 in radians.
% The color of the displayed arch is prescribed by ring_color, a 3-vector encoding the rgb tr
```

```
% Build arch in .01 radian increments, being sure to include the endpoint
```

```

theta = [theta1:.01:theta2 theta2];
n = length(theta);

% Counter-clockwise around the outside

outx = a + r2*cos(theta);
outy = b + r2*sin(theta);

% Clockwise around the inside

inx = a + r1*cos(theta(n:-1:1));
iny = b + r1*sin(theta(n:-1:1));

% Draw the ring

fill([outx inx], [outy iny], ring_color);

http://www.math.berkeley.edu/~hthall/128A/matlab/OlympicRings.m

function OlympicRings(r, n, ring_colors)

% Draws n olympic-style rings of radius r, with colors
% specified by the rows of the n-by-3 matrix ring_colors

% find the centers
x = [0:n-1]*r*1.15;
y = zeros(1,n); y([2:2:n]) = -r;

% Preserve the status of "hold" in the long term, but hold for now
t_hold = ishold;
hold on;

% Fill in the upper crossings left to right . . .
% left and right quadrants of odd numbered (upper) rings
% top and bottom quadrants of even numbered (lower) rings
for k=1:n
    Arch(x(k),y(k),(1/4+k/2)*pi, (3/4+k/2)*pi, .85*r, r, ring_colors(k,1:3));
    Arch(x(k),y(k),(5/4+k/2)*pi, (7/4+k/2)*pi, .85*r, r, ring_colors(k,1:3));
end

% And the lower crossings right to left!
% top and bottom quadrants of odd numbered (upper) rings
% left and right quadrants of even numbered (lower) rings
for k=n:-1:1
    Arch(x(k),y(k),(3/4+k/2)*pi, (5/4+k/2)*pi, .85*r, r, ring_colors(k,1:3));
    Arch(x(k),y(k),(7/4+k/2)*pi, (9/4+k/2)*pi, .85*r, r, ring_colors(k,1:3));
end

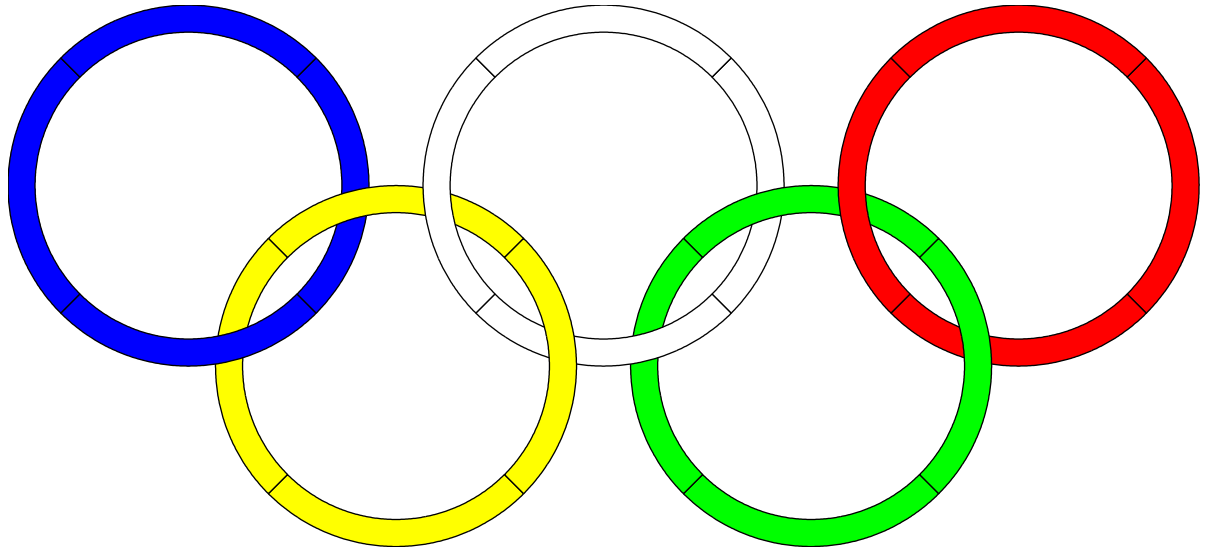
```

```

end

if t_hold == 0
    hold off
end

```



(4) Derive running error bounds that can be evaluated concurrently with the sum $S_N = \sum_{j=1}^N a_j$ and give a tight and rigorous bound for the accumulated roundoff error in $\text{fl}(S_N)$.

At each step we accumulate an error of size $(S_{j-1} + a_j)\delta_j$, where S_{j-1} is the partial sum resulting from previous calculations and $\delta_j \leq \epsilon$. Thus we get

$$\begin{aligned}
 \hat{S}_N &= a_1(1 + \delta_2)(1 + \delta_3) \dots (1 + \delta_N) \\
 &+ a_2(1 + \delta_2)(1 + \delta_3) \dots (1 + \delta_N) \\
 &+ a_3(1 + \delta_3)(1 + \delta_4) \dots (1 + \delta_N) \\
 &\dots \\
 &+ a_N(1 + \delta_N)
 \end{aligned}$$

The absolute error is thus bounded by $\epsilon \sum_{j=1}^N 2(N - j + 1)|a_j|$ for reasonable values of ϵ and N .

Of course, this is assuming that the terms a_j are given to us as exact values. If instead they are the result of other calculations, then the roundoff error from those calculations should also be included in the running error bounds, as we do below.

(5) Modify the script `ExpTaylor` on page 42 of Van Loan to plot the running error bounds you derived in part (4), together with the actual error. Find a number N of terms in the Taylor series for $\exp(10)$ which gives full double-precision accuracy in exact arithmetic. Plot the errors and bound for computing $\exp(10)$ and $\exp(-10)$ with N terms in Matlab's arithmetic, computing each sum in both forward and

reverse order. Explain the results in each of the four cases with a roundoff error analysis.

First, to come up with a reasonable N , we approximate the tail of the series with a geometric series. If $N/10$ is at least 2, then each term in the series

$$\sum_{j=N+1}^{\infty} \frac{10^j}{j!}$$

will be less than half the previous term, and so the sum of all the terms we exclude will be less than the last term we do include, namely $\frac{10^N}{N!}$. We're looking for a number $N > 20$ such that

$$\frac{10^N}{N!e^{10}} < 10^{-16},$$

and a little experimentation reveals that $N = 50$ will do.

```

http://www.math.berkeley.edu/~hthall/128A/matlab/ExpTaylor2.m

% Script File: ExpTaylor2.m
% Plots, as a function of n, the relative error in the
% Taylor approximation
%
%           1 + x + x^2/2! +...+ x^n/n!
%
% to exp(x), together with running bounds on the error,
% for x=10 and x=-10, running both backwards and forwards.

N = 51; % zero to 50

% Pre-compute the terms of the Taylor series, along with
% a bound for the absolute error in each term.
%
% For exp(-10), we'll just negate the odd-order terms
% (and keep the same error bound); two more columns
% will be the same thing in the reverse order.

term = zeros(N,4); % [10fwd, -10fwd, 10rev, -10rev]
termErr = zeros(N,4);
DELTA = 0;

% Easiest to do -10fwd first and then take absolute values

thisterm = 1;
for k=1:N
    term(k,2) = thisterm;
    termErr(k,2) = abs(thisterm*DELTA); % 0 term is exact

    thisterm = - thisterm * 10;
    DELTA = DELTA + eps; % = 2*eps/2

```

```

    thisterm = thisterm / k;
    DELTA = DELTA + eps;
end

% Fill in terms for 10fwd
% term(:, 1) means the entire 1st column

term(:, 1) = abs(term(:, 2));
termErr(:, 1) = termErr(:, 2);

% Reverse order

term(:, 3:4) = term(N:-1:1, 1:2);
termErr(:, 3:4) = termErr(N:-1:1, 1:2);

s = zeros(N,4);      % four sets of partial sums
sumErr = zeros(N,4); % running absolute error bounds
actErr = zeros(N,4); % actual relative error
                    % from partial sum and roundoff

s(1, :) = term(1, :);
f = exp([10 -10 10 -10]);

for k=2:N % 1st row already taken care of
    s(k, :) = s(k-1, :) + term(k, :);
    sumErr(k,:) = (sumErr(k-1, 1:4) + termErr(k, 1:4)) * (1+eps) ...
        + (abs(s(k, :)) + abs(term(k, :))) * eps/2;
    actErr(k, :) = abs(s(k, :) - f)./f;
end

% Convert absolute error bounds to relative error bounds

sumErr = sumErr./ (abs(s) - sumErr);
% This is assuming abs(sum) > sumErr, which might not
% always be true. (For example, in exp(-1), the partial
% sum 1 -1 is zero.)

titles = [' exp(10), forward '
         ' exp(-10), forward'
         ' exp(10), backward'
         'exp(-10), backward']; % matrix of characters

for k=1:4
    subplot(2,2,k);

    % The 'actual error' includes the error from only summing
    % a few terms of the Taylor series, which is much larger
    % than the running error bounds on the accuracy of the

```

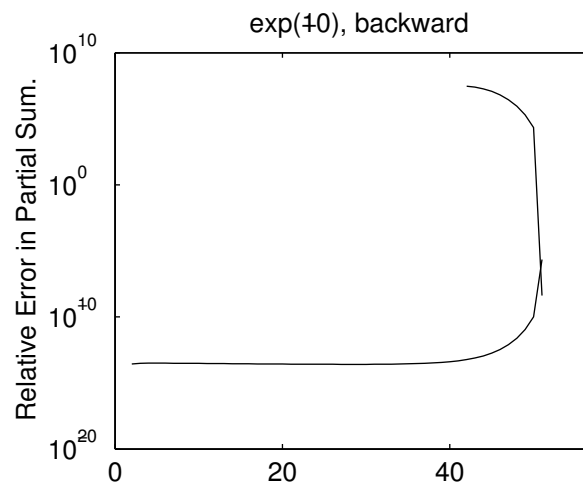
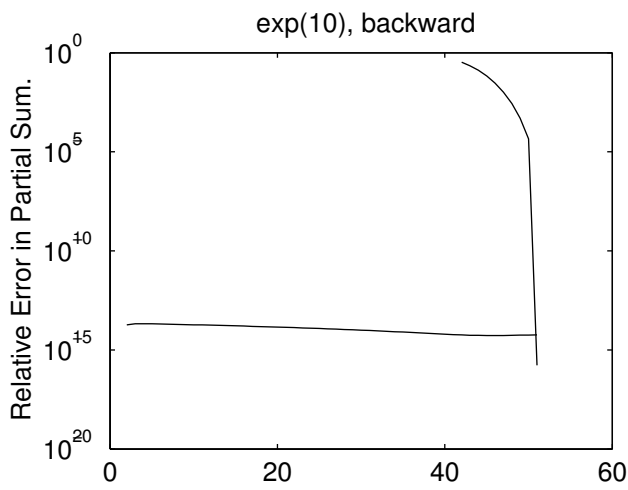
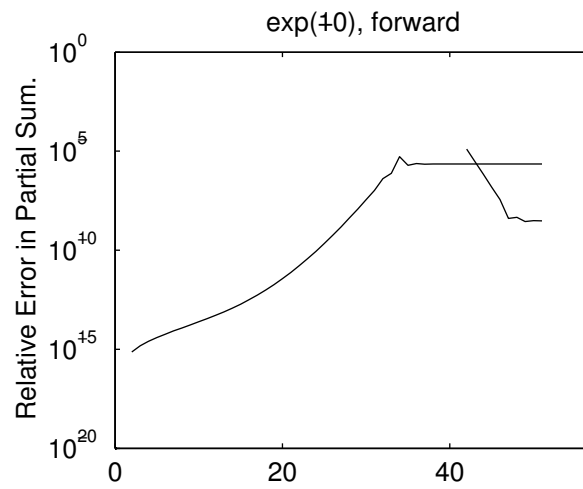
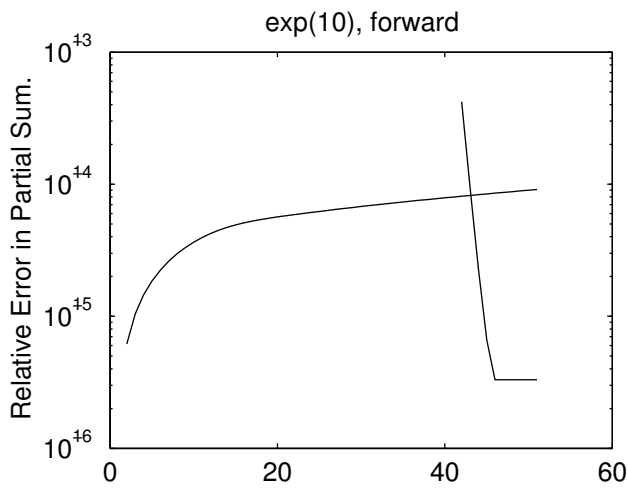
```

% partial sum until the end of the summation.

% In order to plot them on the same scale, here we only
% plot the last 10 terms of the 'actual error'.

semilogy(2:N,sumErr(2:N,k) ,N-9:N,actErr(N-9:N,k))
ylabel('Relative Error in Partial Sum.')
title(titles(k, :))
end

```



<http://www.math.berkeley.edu/~hthall/128A/matlab/ExpTaylorResult.txt>

```
>> [ sumErr(51, [1 3])
```

```
actErr(51, [1 3]) ] % bounds and actual errors for full summation of exp(10)

ans =

1.0e-014 *

0.91038187211170    0.57731597280509
0.03303279646387    0.01651639823194

>> [ sumErr(51, [2 4])
actErr(51, [2 4]) ] % bounds and actual errors for full summation of exp(-10)

ans =

1.0e-005 *

0.22353626330973    0.22353675244730
0.00030597652078    0.00041130816362
```