# Computer Assignment 1 – Solutions

1. a) Matlab m-file `naivequad.m`

```
function x = naivequad(a, b, c)
% Solve quadratic equation a*x^2 + b*x + c = 0
  D = sqrt(b^2 - 4*a*c);
  x = [-b + D; -b - D]/(2*a);
end
```

b) Matlab m-file `robustquad.m`

```
function x = robustquad(a, b, c)
% Solve quadratic equation a*x^2 + b*x + c = 0 using formulas
% that avoid subtractive cancellation
  D = sqrt(b^2 - 4*a*c);
  if b > 0
    x = [-(2*c)/(b+D); -(b+D)/(2*a)];
  else
    x = [-(b-D)/(2*a); -(2*c)/(b-D)];
  end
end
```

c) The subtractive cancellation occurs when $|b| \gg |ac|$, so that $D \approx |b|$. For example (see Exercises 2.2 and 2.3 on p. 24 of Pav's lecture notes),

```
>> format short g
>> x = naivequad(1, 1e15, 1)
x =
           0
      -1e+15
>> x = robustquad(1, 1e15, 1)
x =
      -1e-15
      -1e+15
```

Example with $b < 0$:

```
>> x = naivequad(1, -1e15, 1)
x =
       1e+15
           0
>> x = robustquad(1, -1e15, 1)
x =
       1e+15
       1e-15
```

Example with marginal loss of precision:

```
>> x = naivequad(1, 1e8, 1)
x =
 -7.4506e-09
      -1e+08
>> x = robustquad(1, 1e8, 1)
x =
      -1e-08
      -1e+08
```

2. a) Matlab m-file `fdiff.m`

```
function fp = fdiff(f, x, h)
%  Compute derivate of f(x) using forward difference formula
  fp = (f(x+h) - f(x))/h;
end
```

Testing the function:

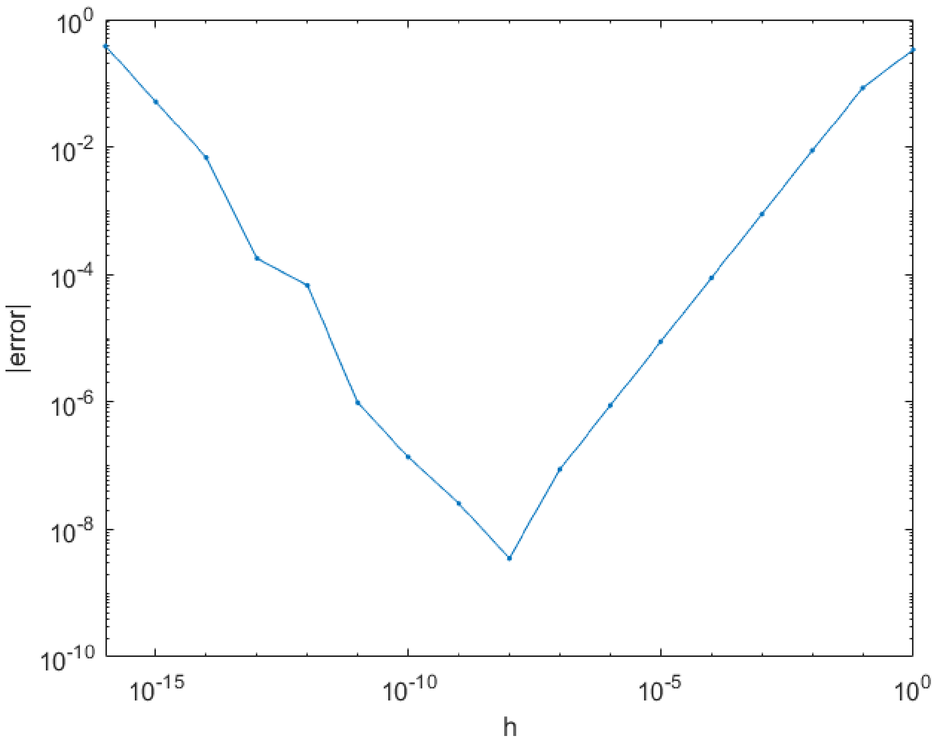
```
>> format long
>> fp = fdiff(@(x)exp(-x.^2), 0.2, 1e-6)
fp =
  -0.384316659607364
```

Compare this to the result from analytic evaluation of the derivative:

```
>> x = 0.2;
>> -2*x*exp(-x.^2)
ans =
  -0.384315775660929
```

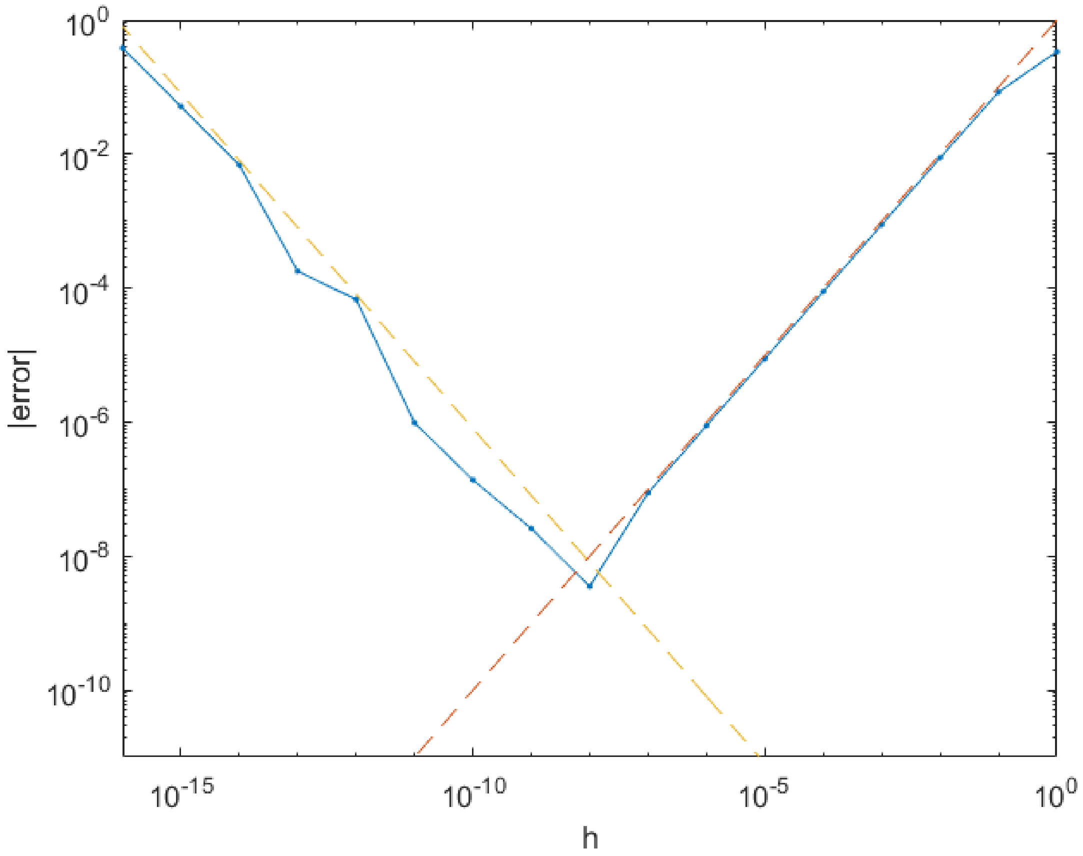
b) Matlab script for computing and plotting the error:

```
x = 0.2;
k = 0:16;  % vector of integers from 0 to 16
for j = 1:length(k)  % loop over values of k
  h(j) = 10^-k(j);   % h = 10^(-k)
  fp(j) = fdiff(@(x)exp(-x^2), x, h(j)); % calculate forward difference
approximation
end
error = abs(fp+2*x*exp(-x^2));
loglog(h,error,'.-');  % plot absolute error vs h on a log-log scale
xlabel('h');  ylabel('|error|');  % axis labels
xlim([1e-16 1]); % adjust the range of x axis
```



c) Script to add lines for estimated truncation and rounding errors:

```
hold on; % keep the previous plot
M = 2.0; ee = 4e-17;
loglog(h,M*h/2,'--',h,2*ee./h,'--');  % plot the truncation error line with
M = 2.0 and rounding error line with e = 4e-17
ylim([1e-11 1]); % adjust the range of the y axis
```




We see that the lines have correct slopes and we choose parameters $M$ and $\epsilon$ so that the lines bound the total error from above.

The value $M = 2.0$ is close to the value estimated from the 2nd derivative of the function at $x = 0.2$:

```
>> syms x
>> fpp = diff(exp(-x^2),2)
fpp =
4*x^2*exp(-x^2) - 2*exp(-x^2)
>> abs(eval(subs(fpp,x,0.2)))
ans =
    1.7679
```

The value $\epsilon = 4 \times 10^{-17}$ is much smaller than Matlab's machine precision value

```
>> eps
ans =
   2.2204e-16
```

This may indicate that Matlab uses additional digits of precision in order to carry out more precise calculations. Note that $\epsilon$ is the absolute error, while indicates relative error. But in this case $\exp(-0.2^2) = 0.9608 \approx 1$, so the comparison is appropriate.