1.(a) (5 points) What makes a quadrature procedure like \texttt{quad} adaptive? Why is this important?

\textit{Solution}

It figures out the subinterval lengths. Shorter lengths where it is suspected that the function is changing rapidly. This reduces the overall number of function evaluations.

1.(b) (5 points) Why is it not advisable to use a high degree polynomial to interpolate a function \(f\)?

\textit{Solution}

The error will depend on the \(f^{(n)}\) if there are \(n\) interpolation points.

1.(c) (5 points) Matrix factorizations are important because they can be used to transform a given problem into an equivalent, easy-to-solve problem. Illustrate this by showing how the QR factorization can be used to minimize \(\|Ax - b\|_2\) where \(A \in \mathbb{R}^{m \times n}\) has rank \(n\) and \(b \in \mathbb{R}^m\).

\textit{Solution}

\[\|Ax - b\|_2 = \|QRx - b\|_2 = \|Q^T(QRx - b)\|_2 = \|Rx - Q^Tb\|_2\]

If \(d = Q^Tb\), then the minimizing \(x\) solves \(R(1:, n, 1:n)x = d(1:n)\).

1.(d) (5 points) Suppose

\[A = USV^T = \sum_{i=1}^{n} S(i, i)U(:, i)V(:, i)^T\]

is the singular value decomposition of a matrix that represents a black-and-white image. How can we obtain a compressed representation of the image using this decomposition?

\textit{Solution:}

Truncate the summation

\[A_{\text{compressed}} = \sum_{i=1}^{r} S(i, i)U(:, i)V(:, i)^T\]

at a point where the singular values become relatively small.

2. Newton’s method and the secant method can each be applied to the problem of finding a zero of a function \(f: \mathbb{R} \to \mathbb{R}\).
(a) (5 points) Draw a picture that depicts a Newton method step. Indicate clearly the location of the next iterate.

Solution:

Draw a curve. Put a single dot on the curve. Draw a tangent line thru the dot. The next iterate is where the tangent line crosses the x-axis.

(b) (5 points) Draw a picture that depicts a secant method step. Indicate clearly the location of the next iterate.

Solution:

Draw a curve. Put two dots on the curve. Draw a line thru the two dots. The next iterate is where the line crosses the x-axis

(c) (5 points) Compare the work-per-step of the two methods.

Solution:

Newton: One f-eval and one f’ eval per step. Secant: On f-eval per step. Derivative evaluation is generally more expensive than function evaluation.

3. (10 points) If f has two continuous derivatives then

\[
\frac{f(x + h) - f(x)}{h} = f'(x) + \frac{f''(\eta)}{2} h
\]

where \( \eta \) is in between \( x \) and \( x + h \). Thus, we would expect the error in the divided difference to decrease with \( h \). However, when we run the script

```matlab
x = 1;
disp(' h error')
disp('--------------------------')
for k=1:16
    h = 1/10^k; D = (sin(x+h)-sin(x))/h; error = abs(D - cos(x));
    disp(sprintf('%5.1e %10.3e',h,error))
end
```

we discover that the error decreases and then increases:

<table>
<thead>
<tr>
<th>h</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0e-001</td>
<td>4.294e-002</td>
</tr>
<tr>
<td>1.0e-002</td>
<td>4.216e-003</td>
</tr>
<tr>
<td>1.0e-003</td>
<td>4.208e-004</td>
</tr>
<tr>
<td>1.0e-004</td>
<td>4.207e-005</td>
</tr>
<tr>
<td>1.0e-005</td>
<td>4.207e-006</td>
</tr>
<tr>
<td>1.0e-006</td>
<td>4.207e-007</td>
</tr>
<tr>
<td>1.0e-007</td>
<td>4.183e-008</td>
</tr>
<tr>
<td>1.0e-008</td>
<td>2.970e-009</td>
</tr>
<tr>
<td>1.0e-009</td>
<td>5.254e-008</td>
</tr>
<tr>
<td>1.0e-010</td>
<td>5.848e-008</td>
</tr>
<tr>
<td>1.0e-011</td>
<td>1.169e-006</td>
</tr>
<tr>
<td>1.0e-012</td>
<td>4.324e-005</td>
</tr>
<tr>
<td>1.0e-013</td>
<td>7.339e-004</td>
</tr>
<tr>
<td>1.0e-014</td>
<td>3.707e-003</td>
</tr>
<tr>
<td>1.0e-015</td>
<td>1.481e-002</td>
</tr>
<tr>
<td>1.0e-016</td>
<td>5.403e-001</td>
</tr>
</tbody>
</table>

It must be that the effect of roundoff error eventually dominates the effect of the “calculus” error. Explain this in a few sentences using the behavior of the function \( e(h) = h + (\text{positive constant})/h \) to back up your argument.
Half Credit:

Any rounding error in the numerator is magnified by $1/h$.

Thus, as $h$ gets smaller, the calculus error decreases while the roundoff error grows. The crossing point is about $10^{-8}$.

Full credit:

If $\text{eps}$ is the unit roundoff, then we can expect the order of magnitude of the total error to be of the form

$$h + \text{eps}/h = h + (10^{-16})/h$$

This function is minimized around $\sqrt{10^{-16}}$.

4. (15 points) Complete the following function so that it performs as specified

```matlab
function ystar = SplineMin(x,y)
% x and y are column n-vectors with x(1) < x(2) < ... < x(n)
% Let S be the cubic spline interpolant of (x(1),y(1)),..., (x(n),y(n))
% ystar is the minimum value of S on the interval [x(1),x(n)]

Your implementation must make effective use of `fminbnd`. Do not approach this problem by looking at zeros of $S'$. Do not worry about the efficiency of the objective function that is passed to `fminbnd`. Explot the fact that $S$ is a piecewise cubic.

Solution:

```matlab
S = spline(x,y);
for i=1:n-1
    xstar(i) = fminbnd(@(t) f(t,S),x(i),x(i+1))
end
ystar = min(ppval(S,xstar));

function alpha = f(t,S)
    alpha = ppval(S,t)
```

Objective function: 7 points
Application of `fminbnd` to each local cubic: 8 pts

- using `polyval(c(i,:),x)` to evaluate the polynomial for $x$ in `[breaks(i), breaks(i+1)]` [-3 points].
- Not doing the final minimization over local minima in a vectorized way. [-1 point]
- Invoking `polyval` on $c(i)$ rather than on $c(i,:)$ where $c$ is the matrix of coefficients. [-3 points]

Code (a) : Not exploiting the *piecewise* structure, i.e. invoking `fminbnd` only once on the interval $x(1),x(n)$ [-10 points]

Code (*) : Arguing that the global minimum should be in one of the two intervals around $x_i$ where $i = \text{argmin}_i (y_i)$ This is wrong.
5. (a) (10 points) Given \(x, y \in \mathbb{R}^4\) and \(a \in \mathbb{R}\), how would you solve the following linear system using \texttt{polyfit}:

\[
\begin{bmatrix}
1 (x_1 - a) & (x_1 - a)^2 & (x_1 - a)^3 & \vdots & c_1 \\
1 (x_2 - a) & (x_2 - a)^2 & (x_2 - a)^3 & \vdots & c_2 \\
1 (x_3 - a) & (x_3 - a)^2 & (x_3 - a)^3 & \vdots & c_3 \\
1 (x_4 - a) & (x_4 - a)^2 & (x_4 - a)^3 & \vdots & c_4 \\
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3 \\
c_4 \\
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
\]

Solution:

\[
\begin{aligned}
b &= \texttt{polyfit}(x-a,y) \\
c &= b(4:-1:1)
\end{aligned}
\]

- Not doing \(c = c(4:-1:1)\) [-3 points]
- Doing it but in a non-vectorized way [-2 points]
- Invoking made up functions or subfuntions like "reverse(v)" [-2 points]

5. (b) (10 points) Suppose \(A \in \mathbb{R}^{m \times n}\), \(x \in \mathbb{R}^n\), and \(y \in \mathbb{R}^m\) are initialized MATLAB variables. Write a MATLAB script that computes

\[
s = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}y_i x_j
\]

Your solution should be vectorized to receive full credit. Hint: Look for a matrix-vector multiplication.

Solution:

\[
s = y' * A * x
\]

- Missing the transpose in \(y\) [-1 point]
- Doing something more contrived like \(\text{sum}(A*x,y)\) [-2 points]
- Doing something that requires more memory like \(\text{sum(sum(A.*(x*y')))\} [-2 points]
- Carrying out one sum only (i.e returning a vector) [-2 points].

6. (20 points) Assume that a function \(F(t,n)\) is available and that it returns a column n-vector whose components are continuous functions of the scalar \(t\). Complete the following function so that it performs as specified.

\[
\text{function Q=G(A,a,b,tol)}
\]

\[
\% A is an n-by-n nonsingular matrix. \\
\% a and b are scalars with a < b. \\
\% tol is a positive number \\
\% Q is an approximation of the integral of } g(t) = F(t,n)'*inv(A)*F(t,n) \\
\% from t=a to t=b obtained by using quad with absolute tolerance tol.
\]

\[
[n,n] = \text{size}(A); \\
[L,U,P] = \text{lu}(A); \\
Q = \text{quad}(\@t f(t,n,L,U,P),a,b,tol)
\]

\[
\text{function alfa = f(t,n,L,U,P)} \\
\text{v = F(t,n); \% v = F(t,n);} \\
\text{alfa = v'*(U\setminus(L\setminus(P*v)));}
\]

Precomputing \(lu\) factorization: 5 points

An integrand function that works: 5 points

Using the \(LU\) factorization correctly: 5 points

Evaluating \(F\) only once in integrand function: 5 points