AM205: Assignment 1 solutions

Problem 1

We are given the data set $x = (0, 1, 2, 3)$ and $y = (0, 0, 1, 2)$. We want to show that using the monomial basis and using the Lagrange basis give us the same polynomial interpolation. We first consider the monomial basis. We want to solve for the interpolation coefficients $v = (a, b, c, d)$ in the form of $y = a + bx + cx^2 + dx^3$. The Vandermonde matrix is given by

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix}. \quad (1)$$

Solving the linear system $Av = y$ as $v = A^{-1}y$ yields

$$v = \left( 0, -\frac{5}{6}, 1, -\frac{1}{6} \right) \quad (2)$$

and hence the polynomial interpolant is $y = -\frac{5}{6}x + x^2 - \frac{1}{6}x^3$. We now consider the Lagrange basis. We interpolate the data using Lagrange polynomials in the form of

$$y(x) = y_0L_0(x) + y_1L_1(x) + y_2L_2(x) + y_3L_3(x), \quad (3)$$

where the Lagrange polynomials are defined as

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}. \quad (4)$$

For the given data, the four polynomials are

$$L_0 = -\frac{1}{6}(x-1)(x-2)(x-3),$$
$$L_1 = \frac{1}{2}(x-0)(x-2)(x-3),$$
$$L_2 = -\frac{1}{2}(x-0)(x-1)(x-3),$$
$$L_3 = \frac{1}{6}(x-0)(x-1)(x-2). \quad (5)$$

After simplification, we obtain $y = -\frac{5}{6}x + x^2 - \frac{1}{6}x^3$, which matches the result for the monomial basis. Figure 1 shows the interpolated function $y(x)$ with the data points superimposed.

*Solutions were written Kevin Chen (TF, Fall 2014), Dustin Tran (TF, Fall 2014), and Chris H. Rycroft. Edited by Chris H. Rycroft.*
Problem 2 – Error bounds with Lagrange Polynomials

Parts (a) and (b)

Figure 2 shows the Lagrange polynomial $p_3(x)$ over the true function $f(x)$ using a slightly modified version of the in-class code example. Running the code, the infinity norm of the error is approximately 5.75191.

Part (c)

The upper bound for the interpolation error is given by

$$
\|f(x) - p_{n-1}(x)\|_\infty = \left\| \frac{f^{(n)}(\theta)}{n!} \prod_{i=1}^{n}(x - x_i) \right\|_\infty, \tag{6}
$$

where $\theta$ is a specific value within the interval from $-1$ to $1$. Applying the Cauchy–Schwarz inequality gives

$$
\|f(x) - p_{n-1}(x)\|_\infty \leq \frac{\|f^{(n)}(\theta)\|_\infty}{n!} \left\| \prod_{i=1}^{n}(x - x_i) \right\|_\infty. \tag{7}
$$
Using the properties of the Chebyshev polynomials, and that the function $f$ achieves its maximum value at $x = 1$,

$$
\|f(x) - p_{n-1}(x)\|_{\infty} \leq \frac{4^n e^{4\vartheta} + (-2)^n e^{-2\vartheta}}{n!} \frac{1}{2^{n-1}}
\leq \frac{4^n e^4 + (-2)^n e^{-2}}{n!} \frac{1}{2^{n-1}}
\leq \frac{2^{n+1} e^4 + (-1)^n 2 e^{-2}}{n!}.
$$

(8)

**Part (d)**

There are many ways to find better control points, and this problem highlights that while the Chebyshev points are a good set of points at which to interpolate a general unknown function, it is usually rather straightforward to find a better set of interpolating points for a specific function.

One simple method is to examine where the maximum interpolation error is achieved. This is between the second and third control points; hence if we move the second control point closer to the right, and/or the third control point closer to the left, $p_3(x)$ would have a better approximation of $f(x)$ within this region.

We need only be careful not to move these control points too close together, or else we would achieve an even larger infinity norm as a result of a different region of $|p_3(x) - f(x)|$. 

Figure 2: The function $f(x) = e^{4x} + e^{-2x}$ and the interpolating polynomial $p_3(x)$ considered in problem 2.
getting larger. This also extends to the realization that in order to minimize the infinity norm, we want the maximum interpolation error to be achieved inbetween each control point equally. In other words, we aim to distribute the error as equally as possible. In this case, we shift the second control point by 0.2, which leads to an infinity norm of 4.76548.

Problem 3 – The condition number

Part (a)
Throughout this equation, $||\cdot||$ is taken to mean the Euclidean norm. The first two parts of this problem can be solved using diagonal matrices only. Consider first

$$B = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

(9)

Then $||B|| = 2$, $||B^{-1}|| = 1$ and hence $\kappa(B) = 2$. Similarly, $\kappa(C) = 2$. Adding the two matrices together gives

$$B + C = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} = 3I$$

(10)

and hence $\kappa(B + C) = ||3I|| ||\frac{1}{3}I|| = 3 \times \frac{1}{3} = 1$. For these choices of matrices, $\kappa(B + C) < \kappa(B) + \kappa(C)$.

Part (b)

If

$$B = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

(11)

then $\kappa(B) = 2$ and $\kappa(C) = 1$. Adding the two matrices together gives

$$B + C = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix}$$

(12)

and hence $\kappa(B + C) = 5$. Therefore $\kappa(B + C) > \kappa(B) + \kappa(C)$.

Part (c)

Let $A$ be an invertible $2 \times 2$ symmetric matrix. First, note that

$$||2A|| = \max_{v \neq 0} \frac{||2Av||}{||v||} = \max_{v \neq 0} \frac{2||Av||}{||v||} = 2 \max_{v \neq 0} \frac{||Av||}{||v||} = 2||A||.$$  

(13)
Similarly, note that \( \| (2A)^{-1} \| = \| \frac{1}{2}A^{-1} \| = \frac{1}{2} \| A^{-1} \| \). Hence
\[
\kappa(2A) = \| 2A \| \| (2A)^{-1} \| = 2 \| A \| \times \frac{1}{2} \| A^{-1} \| = \| A \| \| A^{-1} \| = \kappa(A).
\] (14)

Now suppose that \( A \) is a symmetric invertible matrix. Then there exists an orthogonal matrix \( Q \) and a diagonal matrix \( D \) such that
\[
A = Q^T D Q.
\] (15)
Since \( Q^T Q = QQ^T = I \), it follows that
\[
A^2 = Q^T D Q Q^T D Q = Q^T D^2 Q.
\] (16)
The matrix norm of \( \| A \| \) is
\[
\| A \| = \max_{v \neq 0} \frac{\| Q^T D Q v \|}{\| v \|}.
\] (17)
Since \( Q \) corresponds to a rotation or reflection, it preserves distances under the Euclidean norm and hence \( \| Q w \| = \| w \| = \| Q^T w \| \) for an arbitrary vector \( w \). Therefore
\[
\| A \| = \max_{v \neq 0} \frac{\| D Q v \|}{\| Q v \|} = \max_{u \neq 0} \frac{\| D u \|}{\| u \|} = \| D \|
\] (18)
where \( u = Q v \). Similarly \( \| A^{-1} \| = \| Q^T D^{-1} Q \| \), and since \( D^{-1} \) is also diagonal it follows that \( \| A^{-1} \| = \| D^{-1} \| \), so \( \kappa(A) = \kappa(D) \). With reference to the condition number notes, \( \kappa(A) = |\alpha \beta^{-1}| \) where \( \alpha \) is the diagonal entry with largest magnitude and \( |\beta| \) is the diagonal entry with the smallest entry with smallest magnitude.

Since \( D^2 \) is also diagonal, it follows that \( \| A^2 \| = \| D^2 \| \). The diagonal entry of \( D^2 \) with the largest amplitude will be \( \alpha^2 \), and the diagonal entry of \( D^2 \) with the smallest amplitude will be \( \beta^{-2} \). Hence
\[
\kappa(A^2) = |\alpha^2 \beta^{-2}| = (\kappa(A))^2.
\] (19)

Part (d)
The result for that \( \kappa(2A) = \kappa(A) \) is true for arbitrary \( 2 \times 2 \) invertible matrices. The derivation that was considered in part (c) did not rely on the matrix being symmetric.

The result about \( \kappa(A^2) \) does not generalize to arbitrary matrices. If
\[
A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}
\] (20)
then
\[
A^2 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}.
\] (21)
One can numerically verify that \( \kappa(A^2) = 5.828 \) but \( (\kappa(A))^2 = 6.854 \), so the two do not agree.
We aim to construct a cubic spline where $\alpha$, $\beta$, $\gamma$, and $\delta$ are chosen because of their favorable properties at $t$ (giving eight constraints). The first and second derivatives must also be continuous at the interface between each cubic, giving another eight constraints. Hence, the number of free parameters and the number of constraints agree, so there should be a unique solution.

A possible pitfall is to try and find the solution using a library routine, such as SciPy’s `interp1d`. However, the boundary conditions considered here are distinctly different from the defaults usually used by these routines. A standard (non-periodic) spline uses the conditions $s_x''(0) = 0$ and $s_x''(4) = 0$. On the other hand, to make the spline periodic, this problem uses the conditions $s_x''(0) = s_x''(4)$ and $s_x'(0) = s_x'(4)$. The different conditions fundamentally alter the solution.

To construct the spline, we make use of a basis of cubics

$$
\begin{align*}
c_0(t) &= t^2(3-2t), \\
c_1(t) &= -t^2(1-t), \\
c_2(t) &= (t-1)^2t, \\
c_3(t) &= 2t^3 - 3t^2 + 1,
\end{align*}
$$

which are chosen because of their favorable properties at $t = 0$ and $t = 1$ that are summarized in Table 1. Using these functions, the spline can be written as

$$
s_x(t) = \begin{cases} 
  c_0(t) + \alpha c_1(t) + \delta c_2(t) & \text{for } t \in [0,1), \\
  c_3(t-1) + \beta c_1(t-1) + \alpha c_2(t-1) & \text{for } t \in [1,2), \\
  -c_0(t-2) + \gamma c_1(t-2) + \beta c_2(t-2) & \text{for } t \in [2,3), \\
  -c_3(t-3) + \delta c_1(t-3) + \gamma c_2(t-3) & \text{for } t \in [3,4),
\end{cases}
$$

where $\alpha$, $\beta$, $\gamma$, and $\delta$ are unknown constants. The contributions of $c_0$ and $c_3$ are chosen to ensure each cubic matches $\sin \frac{\pi t}{2}$ at the end points. The pairwise occurrences of each constant are chosen to ensure that the first derivative is continuous.

To set the free parameters, the second derivatives must be made continuous at each interface between each cubic, giving another eight constraints. Hence, the number of free parameters and the number of constraints agree, so there should be a unique solution.

A possible pitfall is to try and find the solution using a library routine, such as SciPy’s `interp1d`. However, the boundary conditions considered here are distinctly different from the defaults usually used by these routines. A standard (non-periodic) spline uses the conditions $s_x''(0) = 0$ and $s_x''(4) = 0$. On the other hand, to make the spline periodic, this problem uses the conditions $s_x''(0) = s_x''(4)$ and $s_x'(0) = s_x'(4)$. The different conditions fundamentally alter the solution.

To construct the spline, we make use of a basis of cubics

$$
\begin{align*}
c_0(t) &= t^2(3-2t), \\
c_1(t) &= -t^2(1-t), \\
c_2(t) &= (t-1)^2t, \\
c_3(t) &= 2t^3 - 3t^2 + 1,
\end{align*}
$$

which are chosen because of their favorable properties at $t = 0$ and $t = 1$ that are summarized in Table 1. Using these functions, the spline can be written as

$$
s_x(t) = \begin{cases} 
  c_0(t) + \alpha c_1(t) + \delta c_2(t) & \text{for } t \in [0,1), \\
  c_3(t-1) + \beta c_1(t-1) + \alpha c_2(t-1) & \text{for } t \in [1,2), \\
  -c_0(t-2) + \gamma c_1(t-2) + \beta c_2(t-2) & \text{for } t \in [2,3), \\
  -c_3(t-3) + \delta c_1(t-3) + \gamma c_2(t-3) & \text{for } t \in [3,4),
\end{cases}
$$

where $\alpha$, $\beta$, $\gamma$, and $\delta$ are unknown constants. The contributions of $c_0$ and $c_3$ are chosen to ensure each cubic matches $\sin \frac{\pi t}{2}$ at the end points. The pairwise occurrences of each constant are chosen to ensure that the first derivative is continuous.

To set the free parameters, the second derivatives must be made continuous at each interface between each cubic, giving another eight constraints. Hence, the number of free parameters and the number of constraints agree, so there should be a unique solution.

A possible pitfall is to try and find the solution using a library routine, such as SciPy’s `interp1d`. However, the boundary conditions considered here are distinctly different from the defaults usually used by these routines. A standard (non-periodic) spline uses the conditions $s_x''(0) = 0$ and $s_x''(4) = 0$. On the other hand, to make the spline periodic, this problem uses the conditions $s_x''(0) = s_x''(4)$ and $s_x'(0) = s_x'(4)$. The different conditions fundamentally alter the solution.

To construct the spline, we make use of a basis of cubics

$$
\begin{align*}
c_0(t) &= t^2(3-2t), \\
c_1(t) &= -t^2(1-t), \\
c_2(t) &= (t-1)^2t, \\
c_3(t) &= 2t^3 - 3t^2 + 1,
\end{align*}
$$

which are chosen because of their favorable properties at $t = 0$ and $t = 1$ that are summarized in Table 1. Using these functions, the spline can be written as

$$
s_x(t) = \begin{cases} 
  c_0(t) + \alpha c_1(t) + \delta c_2(t) & \text{for } t \in [0,1), \\
  c_3(t-1) + \beta c_1(t-1) + \alpha c_2(t-1) & \text{for } t \in [1,2), \\
  -c_0(t-2) + \gamma c_1(t-2) + \beta c_2(t-2) & \text{for } t \in [2,3), \\
  -c_3(t-3) + \delta c_1(t-3) + \gamma c_2(t-3) & \text{for } t \in [3,4),
\end{cases}
$$

where $\alpha$, $\beta$, $\gamma$, and $\delta$ are unknown constants. The contributions of $c_0$ and $c_3$ are chosen to ensure each cubic matches $\sin \frac{\pi t}{2}$ at the end points. The pairwise occurrences of each constant are chosen to ensure that the first derivative is continuous.

To set the free parameters, the second derivatives must be made continuous at each interface between each cubic, giving another eight constraints. Hence, the number of free parameters and the number of constraints agree, so there should be a unique solution.
interface. At \( t = 1, 2, 3, 4 \), by reference to Tab. 1, this gives

\[
-6 + 4\alpha + 2\delta = -6 - 2\beta - 4\alpha, \quad (24)
\]

\[
6 + 4\beta + 2\alpha = -6 - 2\gamma - 4\beta, \quad (25)
\]

\[
6 + 4\gamma + 2\beta = 6 - 2\alpha - 4\beta, \quad (26)
\]

\[-6 + 4\delta + 2\gamma = 6 - 2\alpha - 4\delta, \quad (27)
\]

respectively. In matrix form, this is

\[
\begin{pmatrix}
8 & 2 & 0 & 2 \\
2 & 8 & 2 & 0 \\
0 & 2 & 8 & 2 \\
2 & 0 & 2 & 8 \\
\end{pmatrix}
\begin{pmatrix}
\alpha \\
\beta \\
\gamma \\
\delta \\
\end{pmatrix}
= \begin{pmatrix}
0 \\
-12 \\
0 \\
12 \\
\end{pmatrix}
\]

which has the unique solution \((\alpha, \beta, \gamma, \delta) = (0, -\frac{3}{2}, 0, \frac{3}{2})\). Hence

\[
s_x(t) = \begin{cases}
c_0(t) + \frac{3}{2}c_2(t) & \text{for } t \in [0, 1), \\
c_3(t - 1) - \frac{3}{2}c_1(t - 1) & \text{for } t \in [1, 2), \\
-c_0(t - 2) - \frac{3}{2}c_2(t - 2) & \text{for } t \in [2, 3), \\
-c_3(t - 3) + \frac{3}{2}c_1(t - 3) & \text{for } t \in [3, 4).
\end{cases} \quad (29)
\]

Figure 3(a) shows a plot of \( s_x(t) \) in comparison to \( \sin \frac{\pi t}{2} \). There is a high level of agreement between the two functions.

Since \( \cos \frac{\pi t}{2} \) is the same as \( \sin \frac{\pi t}{2} \), but shifted by \(-1\), it follows that it can be approximated by the spline

\[
s_y(t) = \begin{cases}
c_3(t) - \frac{3}{2}c_1(t) & \text{for } t \in [0, 1), \\
-c_0(t - 1) - \frac{3}{2}c_2(t - 1) & \text{for } t \in [1, 2), \\
-c_3(t - 2) + \frac{3}{2}c_1(t - 2) & \text{for } t \in [2, 3), \\
c_0(t - 3) + \frac{3}{2}c_2(t - 3) & \text{for } t \in [3, 4). \\
\end{cases} \quad (30)
\]

Figure 3(b) shows a plot of \( s_y(t) \) in comparison to \( \cos \frac{\pi t}{2} \).

**Part (d)**

Figure 4(a) shows a plot of the parametric curve \( s(t) = (s_x(t), s_y(t)) \) compared to a true circle. The `am205_s011_ex3.py` code integrates the area enclosed by the this curve by dividing it into a large number of triangles. Recall that if \( \mathbf{a} \) and \( \mathbf{b} \) are vectors along two sides of a triangle, then the area is given by \( \frac{1}{2} |\mathbf{a} \times \mathbf{b}| \). For the given problem, consider the triangle with vertices at the origin, \( s(t) \), and \( s(t + \Delta t) \). Then for \( \Delta t \) small, vectors along two
sides are given by \( s(t) \) and \( s'(t) \Delta t \). Hence the total enclosed area is the sum of triangles such as this, and is hence given by the integral

\[
A = \frac{1}{2} \int_0^4 |s(t) \times s'(t)| \, dt.
\] (31)

The program shows that \( A = 3.05000 \) to five decimal places using Python’s integration routine. The integrand is a fifth order polynomial, and hence it can also be evaluated exactly using alternative means to determine that \( A = \frac{61}{20} \). This differs from \( \pi \) by 3%.

**Part (e)**

Consider calculating a spline \( s^n_x(t) \) of \( \sin \frac{\pi t}{2} \) on the periodic interval \([0, 4]\) using \( n \) control points at \( t = t_0, t_1, \ldots, t_{n-1} \) where \( t_k = \frac{4k}{n} \). By extending the argument from part (a), one can determine that in the interval \([t_k, t_{k+1})\), the spline is given by

\[
s^n_x(t) = (\sin \frac{2k\pi}{n}) c_3(t_*) + (\sin \frac{2(k+1)\pi}{n}) c_0(t_*) + \alpha_{k+1} c_1(t_*) + \alpha_k c_2(t_*)
\] (32)

where \( t_* = \frac{n}{4} (t - t_k) \) and the \( \alpha_k \) are constants for \( k = 0, \ldots, n - 1 \). Due to the periodicity, \( \alpha_{n+k} \) is treated as equivalent to \( \alpha_k \). As in part (a), this form will satisfy that \( s^n_x(t) \) match the
Figure 4: The parametric curve \( s(t) = (s_x(t), s_y(t)) \) compared to a circle, for (a) four control points and (b) seven control points.

function values at the end of each interval, and will have continuous first derivative. To set the \( \alpha_k \), the second derivatives must be considered. Ensuring continuity of the second derivative at \( t_k \) yields

\[
-6 \sin \frac{2k\pi}{n} + 6 \sin \frac{2(k+1)\pi}{n} - 2\alpha_{k+1} - 4\alpha_k = 6 \sin \frac{2(k-1)\pi}{n} - 6 \sin \frac{2k\pi}{n} + 4\alpha_k + 2\alpha_{k-1}.
\] (33)

and hence

\[
2\alpha_{k-1} + 8\alpha_k + 2\alpha_{k+1} = -6 \sin \frac{(k-1)\pi}{2n} + 6 \sin \frac{(k+1)\pi}{2n}.
\] (34)

By considering Eq. 34 for \( k = 0, 1, \ldots, n - 1 \), a linear system is obtained for the \( \alpha_k \), which is the generalization of Eq. 28. The matrix is a circular matrix, where the only entries are \( (2, 8, 2) \) on diagonal lines that wrap around. Linear systems involving such matrices can be solved very rapidly in principle, by employing the fast Fourier transform. Once the \( \alpha_k \) are determined, Eq. 32 gives the explicit form of the spline.

In a similar manner, a spline \( s_y^n(t) \) of \( \cos \frac{n\pi t}{2} \) can be constructed. In part (c), this spline was constructed by noting that this function is the same as sine, but shifted by \(-1\), which permitted the form of \( s_y^n(t) \) to be written down immediately. However, here the same
approach will not always work, since for a general $n$ the positions of the control points may not precisely match when shifted by $−1$. It is therefore necessary to explicitly construct this spline, by using Eqs. 32 and 34 again, and replacing all instances of sine by cosine.

The program `am205_soll_ex3e.py` constructs the two splines for an arbitrary number of control points $n$, when $n \geq 3$. Figure 4(b) shows a plot of the resultant circle approximation using seven points, which becomes near-indistinguishable from a true circle. The program also considers a range of different $n$ and computes the approximation to $\pi$. The area integration is performed using three-point Gaussian quadrature on each subinterval, which results in an exact answer up to truncation error. Figure 5 shows a plot of the relative error $E_{rel}$ in the approximation of $\pi$ as a function of $n$. The relative error is well-fit by the line $E_{rel} = 4.547n^{-4.007}$ over the range $10 < x < 2000$. It is reasonable for the error to scale like fourth power of $n$. If sine and cosine are being well-fit by cubic polynomials, then the leading term in the approximation error will be quartic in size. For $x > 2000$, truncation error becomes visible, as expected.

---

†This will be explained in detail in Unit 3.
Problem 5 – Image reconstruction using low light

Part (a)

In this question we are given a regular $M \times N$ photo of a still-life scene, plus three low-light photos of the same scene that are illuminated in red, green, and blue. Each pixel in the images can be represented as a three-component vector $p = (R, G, B)$ for the red, green, and blue components. Let $p^A_k$ be the $k$th pixel of the regular photo, and let $p^B_k$, $p^C_k$, and $p^D_k$ be the $k$th pixel of the three low-light photos. Here, $k$ is indexed from 0 up to $MN - 1$.

The question requires fitting a model for a regular photo pixel $p^A_k$ in terms of the corresponding low-light pixels by minimizing

$$S = \frac{1}{MN} \sum_{k=0}^{MN-1} ||F^B p^B_k + F^C p^C_k + F^D p^D_k + p_{\text{const}} - p^A_k||^2_2.$$  \hfill (35)

where $F^B$, $F^C$, and $F^D$ are $3 \times 3$ matrices and $p_{\text{const}}$ is a vector. Note that this is equivalent to minimizing three separate quantities in each of the channels,

$$S = S_R + S_G + S_B.$$  \hfill (36)

Here

$$S_R = \frac{1}{MN} \sum_{k=0}^{MN-1} \left( F^{BR} p^B_k + F^{CR} p^C_k + F^{DR} p^D_k + R_{\text{const}} - R^A_k \right)^2.$$ \hfill (37)

where $F^{BR}$, $F^{CR}$, and $F^{DR}$ are the rows of the $F$ matrices corresponding to the red channel. Analogous expressions exist for the green and blue channels. Equation 37 is now a standard linear least squares problem for the ten parameters making up $F^{BR}$, $F^{CR}$, $F^{DR}$, and $R_{\text{const}}$. The program pho.recons.py solves this least squares problem, along with the analogous problems for the green and blue channels. It then uses the fitted parameters to make a reconstruction of the original photo using the three low light images.

Figure 6 shows a comparison between the original photo, and the one that is reconstructed using the low light images. Assume that the pixel values cover the range from 0 to 1. To properly visualize the reconstructed image, the pixel colors are truncated according to $(R, G, B) \rightarrow (T(R), T(G), T(B))$ where $T$ is defined by

$$T(x) = \begin{cases} 
0 & \text{if } x < 0, \\
 x & \text{if } 0 \leq x < 1, \\
 1 & \text{if } x \geq 1. 
\end{cases}$$  \hfill (38)

Overall, the general match of colors in the scene is very good. There are some discrepancies due to differences in lighting. To visualize the differences in more detail, the pixel differences $\Delta p_k = p^A_k - p^A_k$ are computed. These values have both negative and positive
components, and to visualize them, the negative parts and positive parts are plotted separately in Fig. 7. The square error per pixel is

\[ S = \begin{cases} 
0.00405 & \text{for } (M, N) = (400, 300), \\
0.00449 & \text{for } (M, N) = (800, 600), \\
0.00485 & \text{for } (M, N) = (1600, 1200).
\] (39)

Overall, these square errors are very small.

**Part (b)**

The program `photo_recons.py` also uses the fitted model from part (a) to perform a reconstruction of a regular image of the still-life bear scene room using three low-light photos. Figure 8 shows a comparison between the original image and the reconstructed one. Again, the colors across the scene look realistic, which is remarkable given that it is based on a previously fitted model, and three images with very different lighting characteristics. The square error per pixel between the original image and the reconstruction is

\[ T = \begin{cases} 
0.00540 & \text{for } (M, N) = (400, 300), \\
0.00517 & \text{for } (M, N) = (800, 600), \\
0.00532 & \text{for } (M, N) = (1600, 1200).
\] (40)

As expected, the square errors are larger than those from part (a), because the previous model is used, instead of finding the least squares fit for this new set of images. Given small variations in lighting and color, it is not surprising that the previous model is slightly more inaccurate when applied to this set of images. Nevertheless, the square error is still very small.

**Problem 6 – Determining hidden chemical sources**

**Part (a)**

The time derivative of \( \rho_c \) is

\[
\frac{\partial \rho_c}{\partial t} = \frac{1}{4\pi b} \left( \frac{-1}{t^2} + \frac{-(x^2 + y^2)}{4bt} \left( \frac{-1}{t^2} \right) \right) \exp \left( \frac{-x^2 + y^2}{4bt} \right) \\
= \frac{x^2 + y^2 - 4bt}{16\pi b^2 t^3} \exp \left( \frac{-x^2 + y^2}{4bt} \right). \tag{41}
\]

The \( x \) derivative of \( \rho_c \) is

\[
\frac{\partial \rho_c}{\partial x} = \frac{-2x}{16\pi b^2 t^2} \exp \left( \frac{-x^2 + y^2}{4bt} \right). \tag{42}
\]
Figure 6: Comparison between the regular photo of the still-life scene (top), and the reconstructed image based on the best fit of the three low-light photos (bottom).
Figure 7: Differences between the reconstructed image and the regular image, given with pixel values $\Delta p_k = p_k^{\text{rec}} - p_k^{\text{reg}}$. Positive and negative color channel components are shown in the top and bottom images, respectively. Channels are scaled up by a factor of 10 to enhance the differences.
Figure 8: Comparison between the regular photo of two bears (top), and the reconstructed image based on three low-light photos and the previously fitted model (bottom).
and the second $x$ derivative is

$$\frac{\partial^2 \rho_c}{\partial x^2} = \frac{x^2 - 2bt}{16\pi b^3 t^3} \exp \left( -\frac{x^2 + y^2}{4bt} \right).$$  \hspace{1cm} (43)$$

By symmetry the second $y$ derivative is

$$\frac{\partial^2 \rho_c}{\partial y^2} = \frac{y^2 - 2bt}{16\pi b^3 t^3} \exp \left( -\frac{x^2 + y^2}{4bt} \right),$$ \hspace{1cm} (44)$$

and hence

$$\nabla^2 \rho_c = \frac{x^2 + y^2 - 4bt}{16\pi b^3 t^3} \exp \left( -\frac{x^2 + y^2}{4bt} \right).$$ \hspace{1cm} (45)$$

Comparing Eqs. 41 and 45 shows that $\partial_t \rho_c = b \nabla^2 \rho_c$ as required.

**Part (b)**

We now consider the case when $b = 1$ and 49 point sources of chemicals are introduced at $t = 0$ with different strengths, on a $7 \times 7$ regular lattice covering the coordinates $x = -3, -2, \ldots, 3$ and $y = -3, -2, \ldots, 3$. The concentration satisfies

$$\rho(x, t) = \sum_{k=0}^{48} \lambda_k \rho_c(x - v_k, t),$$  \hspace{1cm} (46)$$

where $v_k$ is the $k$th lattice site and $\lambda_k$ is the strength of the chemical introduced at that site.

Two hundred measurements, $\rho_M(x_i, t)$, at locations $x_i$ and at $t = 4$ are provided. Estimating the concentrations can be viewed as a linear least squares problem, finding the $\lambda_k$ such that

$$S = \sum_{i=0}^{199} \left| \rho_M(x_i, t) - \sum_{k=0}^{48} \lambda_k \rho_c(x_i - v_k, t) \right|^2.$$ \hspace{1cm} (47)$$

Even though Eq. 47 is quite complicated and involves the the expression for $\rho_c$, the parameters $\lambda_k$ still enter linearly, and hence it can be solved using the linear least squares approach. The program `solve_cons.py` computes the $\lambda_k$ and prints them. They are all positive, with a maximum value of approximately 252.

<table>
<thead>
<tr>
<th>Lattice site</th>
<th>(0, 0)</th>
<th>(0, 1)</th>
<th>(0, 2)</th>
<th>(0, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>St. Dev. of $\lambda_k$</td>
<td>20222</td>
<td>15637</td>
<td>6835</td>
<td>1323</td>
</tr>
</tbody>
</table>

Table 2: Standard deviations in the $\lambda_k$ when the measured concentrations are perturbed by a small normally distributed shift with mean 0 and variance $10^{-8}$. The values are computed based on $N = 65536$ random samples.
Part (c)

The program `stddev_cons.py` performs a sample of $N$ computations of the $\lambda_k$ when each of the $\rho_M$ are perturbed by a small normally distributed shift with mean 0 and variance $10^{-8}$. For each $\lambda_k$, the standard deviation is computed. Due to the sensitivity of the fitting procedure, these small differences in the measurements cause large alterations in the $\lambda_k$. Table 2 shows the standard deviations for the $\lambda_k$ for four lattice sites, which show much larger variations than the actual $\lambda_k$ values that were measured in part (b). The largest errors are at the central $(0, 0)$ lattice site, which is reasonable since it is furthest away from any of the measurements in the file, thus making it most difficult to estimate.