

# Hundred Digit Challenge Solutions

Eric Dussaud, Chris Husband, Hoang Nguyen,  
Daniel Reynolds and Christiaan Stolk \*

May 16, 2002

Our group was formed as a collection of graduate students and one post-doc from the Computational and Applied Math Department at Rice University. For nearly all the problems, we had at least two distinct approaches that gave similar results. Our solutions are given in Table 1 to a level we are confident of accuracy. For the problems marked with a † to the right of the second column, the numerical methods described below may yield much greater accuracy; we give only 30 digits.

Problem	Solution	
1	0.323367431677778761399	
2	0.995262919443354160890311809426	†
3	1.274224152821	
4	-3.30686864747523728007611377089	†
5	0.2143352345	
6	0.0619139544739909428481752164732	†
7	0.725078346268401167468687719251	†
8	0.424011387033688363797433668593	†
9	0.7859336743503714545652	
10	$3.83758797925122610340713318620 \times 10^{-7}$	†

Table 1: Rice solutions to the Hundred Digit Challenge

## Problem 1

### Solution Method 1 by Christiaan Stolk

Make a transformation of variables  $y = -\log x$ . The integral becomes

$$\int_0^{\infty} \cos(ye^y) dy.$$

This integral can be evaluated by doing explicit numerical integration on shorter and shorter subintervals, and then adding up the results. This yields 0.323367431677778. For higher precision this becomes very expensive. Averaging over one or a few oscillations, and then integrating the average function can yield improvement within reasonable time; we obtained the result 0.323367431677778761399. Better results might also be obtained by transforming to  $z = x^{-1} \log x$ , and then using a partial integration trick for oscillatory integrals like in my solution to problem 9.

---

\*Department of Computational and Applied Mathematics, Rice University, Houston, TX, USA.  
Email: dussaud@caam.rice.edu, chusband@caam.rice.edu, hnguy@caam.rice.edu, reynoldd@caam.rice.edu, cstolk@caam.rice.edu.

## Solution Method 2 by Hoang Nguyen

We used the transformation  $w = \log(x)$  to turn the problem into

$$\int_{-\infty}^0 \cos(w e^{-w}) dw$$

The integrand is oscillatory with rapidly decreasing period as  $w$  approaches  $-\infty$ . This was handled by integrating over one cycle at a time. We first computed a list of points where the integrand reaches a minimum of  $-1$ . This was done by solving the nonlinear equations using Newton's method. The integral was computed for each cycle using `quad1` in Matlab. By adding up enough cycles, we got the answer of 0.3233674316.

## Problem 2

### Solution Method 1 by Hoang Nguyen

We computed the position where the photon hits each mirror surface. This involved solving a quadratic equation and picking the correct root. The normal vector to the mirror surface at this location was computed, then the new trajectory was found by reflecting the old trajectory across the normal. These calculations were done using vector arithmetic (no angles or trigonometric functions were used). This was done one mirror at a time until the total distance traveled by the photon equaled 10. The final answer, obtained using Matlab variable precision arithmetic (128-digits), was 0.99526291944335416. A plot of this trajectory is given in Figure 1.

### Solution Method 2 by Eric Dussaud

A photon moving at speed 1 in the  $x$ - $y$  plane starts at  $t = 0$  at  $(x, y) = (0.5, 0.1)$  heading due east. Around every integer lattice point  $(i, j)$  in the plane, a circular mirror of radius  $\frac{1}{3}$  has been erected. We wish to determine the distance of the photon to the origin of the lattice at time  $t = 10$ . Since the photon has speed 1, the total distance traveled from its initial position at  $t = 10$  is 10. Therefore, it suffices to keep track both of the distance travelled by the photon, and of its position, to solve the problem. The difficulty obviously resides in the fact that the photon is likely to reflect several times between  $t = 0$  and  $t = 10$ .

We solved this problem using the following procedure. Starting with the initial position and direction of the photon, we determine an equation for its path, look for the mirror it will first reach (if any), determine the coordinates of intersection of its path with the mirror's surface (this includes choosing the "right" intersection), compute the equation of its new (reflected) path (including direction), and so on. The distance traveled is computed between any two reflections and the procedure is stopped when we have reached  $d = 10$ . Note that the mirrors on which the photon is reflected during its course are identified in a purely empirical manner (that is, taking a guess by plotting its trajectory in the lattice, and checking that the quadratic equation used to compute the intersection coordinates does indeed yield real roots). The different steps involved are summarized in the following paragraphs. We use the following notation:

- $y = k_1x + p_1$  for the equation of the trajectory of the photon before it reflects on a mirror ( $k_1$  and  $p_1$  are the slope and  $y$ -intercept, respectively),
- $y = k_2x + p_2$  for the equation of the trajectory after reflection,
- $(x - x_0)^2 + (y - y_0)^2 = r^2$  for the equation of the mirror's surface (circle), with  $(x_0, y_0)$  the coordinates of its center, and  $r = \frac{1}{3}$  its radius,

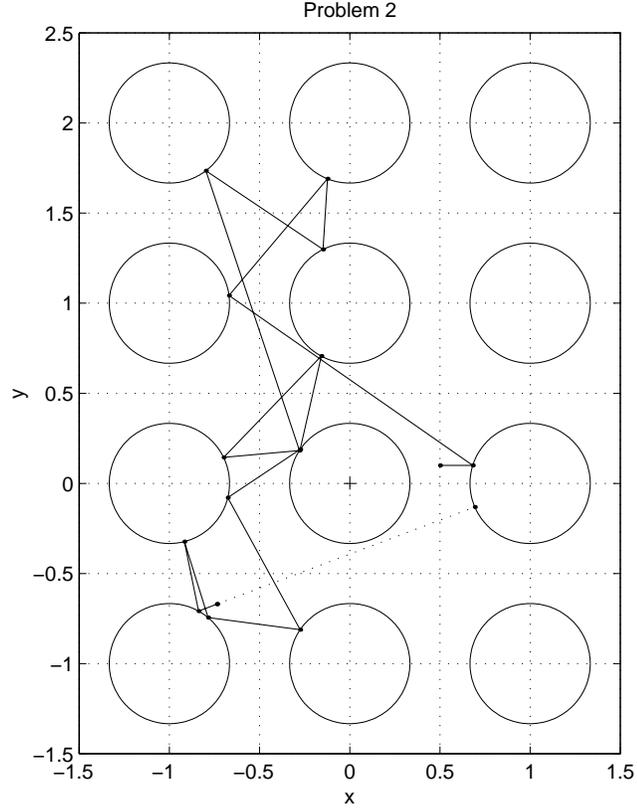


Figure 1: Photon trajectory for Problem 2

- $(x_i, y_i)$  for the coordinates of the intersection of the trajectory with the mirror,
- $a_2x + b_2y + c_2$  for the equation of the normal to the mirror at  $(x_i, y_i)$ .

### Intersection of a line with a circle

The  $x$ -coordinates of the two intersection points satisfy:

$$ax^2 + bx + c = 0,$$

where

$$\begin{aligned} a &= (1 + k_1^2) \\ b &= 2(k_1p_1 - k_1y_0 - x_0) \\ c &= x_0^2 + y_0^2 + p_1^2 - r^2 - 2y_0p_1. \end{aligned}$$

To minimize both *overflow* and *underflow* of the floating point calculations, we used the following procedure to compute the roots  $x_1$  and  $x_2$  of this quadratic equation:

$$\begin{aligned} \Delta &= b^2 - 4ac, \quad q = \frac{-(b + \text{sign}(b)\sqrt{\Delta})}{2} \\ x_1 &= \frac{q}{a}, \quad x_2 = \frac{c}{q}. \end{aligned}$$

The “correct”  $x$ -coordinate  $x_i$  is that for which the distance between the previous reflection point and this new one is minimum . The corresponding  $y$ -coordinate  $y_i$  is simply computed using  $y_i = k_1 x_i + p_1$ , the equation for the photon’s trajectory before reflection.

### Equation for the reflected trajectory

To compute the slope of the reflected trajectory, the equation of the normal to the circle (mirror) at  $(x_i, y_i)$  is used (note that the equation of the tangent could just as well be used). Using the above notation, we have that:

$$a_2 = y_i - y_0, \quad b_2 = x_0 - x_i.$$

Now, the angle  $\alpha$  between the incident trajectory and the normal satisfies:

$$\tan \alpha = \frac{-k_1 b_2 - a_2}{-k_1 a_2 + b_2}.$$

We can compute  $\tan 2\alpha$  from the following trigonometric formula:

$$\tan 2\alpha = \frac{2 \tan \alpha}{1 - \tan^2 \alpha}.$$

Thus, the slope  $k_2$  of the reflected trajectory may be computed as follows:

$$k_2 = \frac{k_1 + \tan 2\alpha}{1 - k_1 \tan 2\alpha}.$$

Its  $y$ -intercept is then computed using the  $(x_i, y_i)$  coordinates

$$p_2 = y_i - k_2 x_i.$$

Note that the above computations do not require evaluating any trigonometric functions.

### Distance

The distances between two points  $(x_1, y_1)$  and  $(x_2, y_2)$  is computed in the usual way:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

### Numerics

All of the numerical computations were performed in *Mathematica*<sup>TM</sup>. We first attempted to obtain each intermediate result in *symbolic* form; after the fourth reflection, the memory required by the symbolic computations was just too large for *Mathematica*<sup>TM</sup> to handle. We started doing numerics at this point. Using the symbolic (exact) expressions obtained thus far for the slopes, intersections, and for the distance traveled, we used variable precision arithmetic (128-digits) to calculate that, at time  $t = 10$ , the photon is at the following distance from the origin:

$$d = 0.995262919443354160890311809426.$$

## Problem 3

### Solution Method 1 by Christiaan Stolk

Let  $A_n$  be the  $n \times n$  matrix obtained by taking the first  $n$  rows and columns of  $A$ . The operator norm  $\|A_n\|$  of  $A_n$  is defined by

$$\|A_n\|^2 = \max_{v \in \mathbb{R}^n} \frac{\|A_n v\|^2}{\|v\|^2}. \quad (1)$$

The problem can be solved by determining the operator norm of  $A_n$  (finding the maximum in (1)) for sufficiently large  $n$ , since  $\|A_n\| \rightarrow \|A\|$  when  $n \rightarrow \infty$ . It is easy to see that all components of the maximizing  $v$  are positive (or all of them negative). Unfortunately, to get the norm to high accuracy,  $n$  must be quite large. (It appears that  $n$  must be exponential in the number of digits, the error is roughly polynomial in  $1/n$ .)

To address this problem we compress the matrix. Let  $Q$  be an orthogonal matrix. Let

$$B = Q^T A_n Q,$$

then  $\|A_n\| = \|B\|$ . Now suppose some rows and columns of  $B$  only have very small entries. Let  $\tilde{Q}$  be the matrix  $Q$  with columns omitted that lead to small entries in  $B$ . To be precise a column  $q_k$  is omitted if  $A_n q_k$  is very small and  $q_k^t A_n$  is also very small. Now let

$$C = \tilde{Q}^T A_n \tilde{Q}.$$

The matrix  $C$  is obtained from  $B$  by removing small rows and columns. Clearly  $\|C\|$  is close to  $\|B\|$ . In the remainder we construct  $\tilde{Q}$  so that the matrix  $C$  is relatively small (approximately 90 by 90 to get 11 digit accuracy for the norm), and the maximization (1) can be done using a standard optimization algorithm.

Let  $V(n)$  be the matrix with  $i$ -th column given by

$$v_i(n) = (0^{i-1}, 1^{i-1}, \dots, (n-1)^{i-1})^T.$$

Let  $W(n)$  be the matrix obtained by doing a Gram–Schmidt orthogonalization of the columns of  $V(n)$ , that is  $w_1(n) = v_1(n)/\|v_1(n)\|$ ,  $w_k(n)$  is a linear combination of the  $v_1(n), \dots, v_k(n)$  that is orthogonal to the  $v_1(n), \dots, v_{k-1}(n)$  and has unit length. Let  $W(n, k)$  be the  $n \times k$  matrix containing the first  $k$  columns of  $W(n)$ .

The entries of the matrix  $A$  are given by

$$A_{i,j} = \left(\frac{1}{2}(i+j-2)(i+j-1) + i\right)^{-1}.$$

Let  $u$  be a part of a row  $u = (A_{i,l+1}, A_{i,l+2}, \dots, A_{i,l+n})$  or of a column  $u = (A_{l+1,j}, A_{l+2,j}, \dots, A_{l+n,j})^T$ . The vector  $w_k(n)$  has  $k-1$  vanishing moments. One can see that if  $l$  is not too small, then the inner product  $u \cdot w_k(n)$  becomes small rapidly when  $k$  increases. This leads to the following definition of  $\tilde{Q}$

$$\tilde{Q} = \begin{pmatrix} I_m & 0 & 0 & \dots & 0 \\ 0 & W(n_1, k_1) & 0 & \dots & 0 \\ 0 & 0 & W(n_2, k_2) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & W(n_K, k_K) \end{pmatrix}.$$

We have chosen  $m = 24$ , and the  $(k_i, n_i)$  equal to  $(16, 40)$ ,  $(12, 64)$ ,  $(12, 128)$ ,  $(10, 256)$ ,  $(7, 512)$ ,  $(5, 1024)$ ,  $(3, 2048)$ ,  $(2, 4096)$ ,  $(1, 8192)$  (in order of increasing  $i = 1, 2, \dots, 9$ ). Thus we obtained a compressed form of  $A_{16384}$ , from which  $\|A\|$  could be computed to approximately 11 digits in Mathematica™. The result was 1.2742241528.

## Solution Method 2 by Hoang Nguyen

The most straightforward approach is to build successively larger matrices in Matlab and use the function `normest` with a very small tolerance parameter. Our hardware was able to handle these dense matrices of order up to about 16000. By studying the convergence rate, we saw that this was more than enough to give at least 10 digits (actually 13 correct digits: 1.274224152821). We could

also have modified the `normest` code to compute matrix-vector products without actually building the matrix. This would allow much larger matrices in case it was necessary. We also tried the Matlab function `norm` to compute the exact 2-norm for matrices of order 10000 or less. This was enough to yield 11 correct digits.

A more elegant approach (by Dr. Yin Zhang, also at Rice University) used the power method and worked on larger matrices. For a matrix of order 100000, we got 1.2742241528212 for the norm. His code is included below.

```
function normA(n)
t0 = cputime;
a = ones(n,1); d = [1:2*n]';
for j = 1:n-1 a(j+1) = a(j) + j; end
x = initialx(n); nrmA = inf;
for iter = 1:10
    b = a; y = sum(x./b)./b;
    for i = 2:n
        b = b+d(i:i+n-1); y = y+sum(x./b)./b;
    end
    nrmA0 = nrmA; nrmA = sqrt(x'*y); x = y/norm(y);
    fprintf(' iter %i: norm(A) = %20.16f\n',iter,nrmA)
    if abs(nrmA0-nrmA) <= 1.e-13*nrmA break; end;
end
fprintf(' CPU time = %g\n',cputime-t0);

function x = initialx(n)
x = zeros(n,1); % make it a better initial guess ...
x(1:58) = [ 0.8440 0.4393 0.2330 0.1403 0.0933 0.0664 0.0497 0.0387...
0.0310 0.0254 0.0212 0.0179 0.0154 0.0134 0.0117 0.0104 0.0092 0.0083...
0.0075 0.0068 0.0062 0.0056 0.0052 0.0048 0.0044 0.0041 0.0038 0.0036...
0.0033 0.0031 0.0029 0.0028 0.0026 0.0024 0.0023 0.0022 0.0021 0.0020...
0.0019 0.0018 0.0017 0.0016 0.0016 0.0015 0.0014 0.0014 0.0013 0.0013...
0.0012 0.0012 0.0011 0.0011 0.0010 0.0010 0.0010 0.0009 0.0009 0.0009];
```

## Problem 4

### Solution Method 1 by Hoang Nguyen

We used a Newton method with line search and a grid of starting points for the global search. The grid increment was 0.01 for both  $x$  and  $y$  directions, but the actual grid values were varied by 0.001 over several runs. For example, the starting point  $(x_0, y_0)$  nearest the origin could be  $(0.000, 0.000)$ ,  $(0.001, 0.001)$ , or  $(-0.001, 0.002)$ , etc. We searched for the smallest local minimum within  $(x, y) \in [-1, 1] \times [-1, 1]$ . We also searched outside this range using a coarser grid of starting points. The best minimum that we found was  $-3.3068686474752$  at  $(x_*, y_*) = (-0.024403079694375, 0.21061242715535)$ .

## Problem 5

### Solution Method 1 by Eric Dussaud

Let  $f(z) = 1/\Gamma(z)$ , where  $\Gamma(z)$  is the gamma function, and let  $p(z)$  be the cubic polynomial that best approximates  $f(z)$  on the unit disk in the supremum norm  $\|\cdot\|_\infty$ . We wish to compute  $\|f - p\|_\infty$  to high accuracy.

The problem is two-fold: the computation of the inverse gamma function itself on one hand, and its cubic approximation on the other hand. In this work, the inverse gamma function is computed via a contour integral due to Hankel [2], [3], and its cubic approximant is taken to be linear combination of Chebyshev polynomials and computed using the software package **COCA** (COmplex Chebyshev Approximation) developed by Bernd Fischer and Jan Modersitzki [1]. It should be noted that none

of the following represents original work: the method for computing the inverse gamma function was taken from the book by Trefethen [3], and the presentation of the mathematical theory behind the COCA package is based entirely on the paper by Fischer and Modersitzki [1].

### Representation of the inverse gamma function

The inverse gamma function may be defined by a contour integral formula due to Hermann Hankel (1864) [2, pp. 234-235]. It takes the form

$$\frac{1}{\Gamma(z)} = \frac{1}{2\pi i} \int_C e^{t-t^z} dt.$$

The  $t$ -plane is cut along the negative real axis, and we write

$$t^{-z} = e^{-z \log t}$$

with the imaginary part of  $\log t$  between  $-\pi$  and  $\pi$ . The contour of integration  $C$  follows the lower edge of the cut from  $-\infty$  to  $-\delta$  ( $\delta > 0$ ), winds around the origin in the positive sense (counterclockwise) on the circle  $|t| = \delta$ , and then follows the upper edge of the cut from  $-\delta$  to  $-\infty$ .

Since the integrand decays exponentially as  $\operatorname{Re}(t) \rightarrow -\infty$ , we can approximate the function as accurately as we like by replacing  $C$  by a bounded contour that begins and ends sufficiently far out on the negative real axis [3]. More specifically, we will take  $C$  to be the circle centered at  $c$  of radius  $r$ . Substituting  $t = c + re^{i\theta}$  in the above formula yields

$$\frac{1}{\Gamma(z)} = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{t-t^z} (t-c) dt.$$

Numerically, the above integral is approximated using the *periodic trapezoid rule*

$$\frac{1}{\Gamma(z)} \approx \frac{1}{N} \sum_{j=1}^N \frac{1}{\Gamma(\theta_j)},$$

with  $\theta_j = j\pi/N$ . Thus, the reciprocal gamma function is approximated by the mean value of  $e^{t-t^z}(t-c)$  over equispaced points on the contour  $C$ . For smooth integrands, it is known that the periodic trapezoid rule converges extraordinarily fast [3].

We note that the inverse gamma function is *analytic* in the complex plane [2]. In particular, it is analytic in the unit disk (a compact set), and continuous on its boundary (the unit circle). Thus the *maximum principle* applies [2]: the reciprocal gamma function attains its maximum modulus on the unit circle. We also note that  $1/\Gamma(z)$  is symmetric with respect to the real axis.

### Cubic approximation of $1/\Gamma(z)$ on the unit disk

We write  $\Omega$  for the open unit disk. Let  $\phi_j$ ,  $j = 1, \dots, n$  be continuous complex-valued functions defined on  $\Omega$ . Then the problem of approximating  $f$  in the Chebyshev sense by a linear combination of the functions  $\phi_j$  may be formulated as follows: determine the coefficients  $c_j \in \mathbb{R}$  (we know the coefficients are real since  $1/\Gamma(z)$  is symmetric with respect to the real axis) such as to minimize

$$\left| f - \sum_{j=1}^n c_j \phi_j \right| = \max_{z \in \Omega} \left| f(z) - \sum_{j=1}^n c_j \phi_j(z) \right|.$$

We choose the functions  $\phi_j$  to be the Chebyshev polynomials of the first kind. Note that these polynomials are continuous in the closed region, and analytic in the open region  $\Omega$ . Thus, the

maximum principle holds, and the minimization problem becomes

$$\min_{c_j \in \mathbb{R}} \left\| f(z) - \sum_{j=1}^n c_j \phi_j(z) \right\|_{\delta\Omega}$$

The above problem was solved using the COCA package, a collection of Matlab functions for computing the best Chebyshev approximation to a function in a setting precisely such as the one we have here. The algorithm implemented in COCA is based on a reformulation of the approximation problem as a semi-infinite optimization problem. The mathematical background of COCA (based on [1]) is presented next.

Let  $\gamma, \gamma([0, 1]) = \delta\Omega$ , denote a piecewise differentiable parametrization of  $\delta\Omega$ . Writing

$$\begin{aligned} \tilde{f}(t) &= f(\gamma(t)), \\ \lambda_j + i\lambda_{j+n} &= c_j, \quad j = 1, \dots, n, \\ \psi_j(t) &= -i\psi_{n+j}(t) = \phi_j(\gamma(t)), \quad j = 1, \dots, n \end{aligned}$$

we can formulate the approximation problem in terms of real coefficients over the (real) interval  $[0, 1]$ :

find coefficients  $\lambda_j^* \in \mathbb{R}^{2n}$  such that

$$h^* = \left\| \tilde{f}(t) - \sum_{j=1}^{2n} \lambda_j^* \psi_j(t) \right\| = \min_{\lambda_j \in \mathbb{R}} \left\| \tilde{f}(t) - \sum_{j=1}^{2n} \lambda_j \psi_j(t) \right\|_{[0,1]}.$$

Let us introduce the error function

$$\epsilon(t; \lambda) = \epsilon(t; \lambda_1, \lambda_2, \dots, \lambda_{2n}) = \tilde{f}(t) - \sum_{j=1}^{2n} \lambda_j \psi_j(t),$$

and the set of extremal points of  $\epsilon(t; \lambda)$

$$E(\lambda) = \{t \in [0, 1] : |\epsilon(t; \lambda)| = \|\epsilon(t; \lambda)\|_{[0,1]}\}.$$

The Chebyshev problem is equivalent to the following nonlinear semi-infinite optimization problem:

$$\begin{aligned} &\text{find the coefficients } \lambda_j^* \in \mathbb{R}^{2n} \text{ so as to} \\ &\quad \text{minimize } h_P \\ &\text{subject to } h_P \geq |\epsilon(t; \lambda)| \text{ for all } t \in [0, 1]. \end{aligned}$$

Next, to obtain a linear problem, the complex modulus in the constraints is replaced by a set of infinitely many linear restrictions

$$|\epsilon(t; \lambda)| = \max_{\alpha \in [-\pi, \pi]} \operatorname{Re}(e^{-i\alpha} \epsilon(t; \lambda))$$

The constraints in the above NLP now read

$$h_P \geq \operatorname{Re}(e^{-i\alpha} \epsilon(t; \lambda)) = \operatorname{Re}(e^{-i\alpha} \tilde{f}(t)) - \sum_{j=1}^{2n} \lambda_j \operatorname{Re}(e^{-i\alpha} \psi_j(t))$$

for all  $t \in [0, 1]$ ,  $\alpha \in [-\pi, \pi]$ . We introduce the following abbreviations:

$$\begin{aligned} \mathbf{a}(t, \alpha) &= [1, \operatorname{Re}(e^{-i\alpha} \psi_1(t)), \dots, \operatorname{Re}(e^{-i\alpha} \psi_{2n}(t))]^T \in \mathbb{R}^{2n}, \\ \mathbf{A}(t, \mathbf{a}) &= (a(t_1, \alpha_1), \dots, a(t_{2n+1}, \alpha_{2n+1})) \in \mathbb{R}^{(2n+1) \times (2n+1)}, \\ c(t, \alpha) &= \operatorname{Re}(e^{-i\alpha} \tilde{f}(t)) \in \mathbb{R}, \\ \mathbf{c}(t, \mathbf{a}) &= (c(t_1, \alpha_1), \dots, c(t_{2n+1}, \alpha_{2n+1}))^T \in \mathbb{R}^{2n+1}. \end{aligned}$$

We can now write the linear semi-infinite problem as

$$\begin{array}{l}
 \text{find the coefficients } \lambda_j^* \in \mathbb{R}^{2n} \text{ so as to} \\
 \text{minimize } h_P \\
 \text{subject to: } \mathbf{a}(t, \alpha) \cdot \begin{pmatrix} h_P \\ \lambda \end{pmatrix} \geq c(t, \alpha) \text{ for all } t \in [0, 1], \alpha \in [-\pi, \pi].
 \end{array}$$

The dual problem is given by

$$\begin{array}{l}
 \text{find points } \mathbf{t} \in [0, 1]^{2n+1}, \text{ angles } \mathbf{a} \in [-\pi, \pi]^{2n+1} \text{ and weights } \mathbf{r} \in [0, 1]^{2n+1} \text{ so as to} \\
 \text{maximize } h_D = \mathbf{c}(\mathbf{t}, \mathbf{a})^T \cdot \mathbf{r} \\
 \text{subject to: } \mathbf{A}(\mathbf{t}, \mathbf{a}) \cdot \mathbf{r} = (1, 0, \dots, 0)^T,
 \end{array}$$

where we used the fact that the best approximation is characterized by  $2n + 1$  points  $t_j, \alpha_j, j = 1, 2, \dots, 2n + 1$ . The primal and dual problems are connected via the so-called characterization theorem, which basically states that the components of the solution vector  $t^*$  of the dual problem are extremal points for the solution of the primal problem. More precisely, COCA solves the following problem:

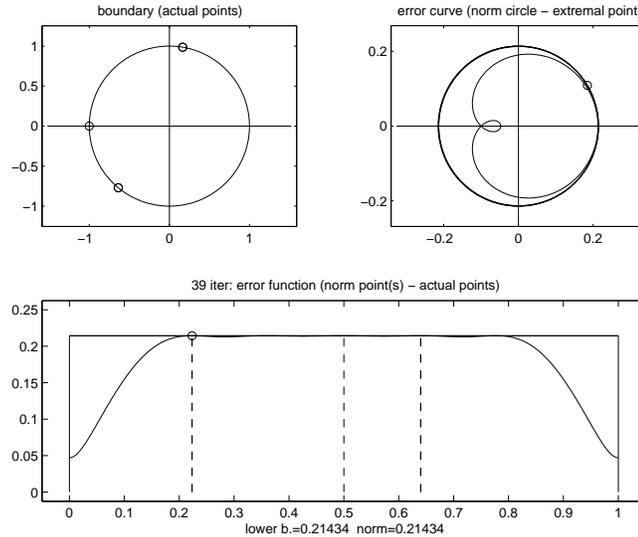


Figure 2: COCA output from Problem 5

$$\begin{aligned}
& \text{find points } \mathbf{t}^* \in [0, 1]^{2n+1}, \text{ angles } \mathbf{a}^* \in [-\pi, \pi]^{2n+1}, \\
& \text{weights } \mathbf{r}^* \in [0, 1]^{2n+1}, \text{ coefficients } \lambda^* \in \mathbb{R}^{2n}, \\
& \text{and a number } h^* > 0 \text{ such that} \\
& \mathbf{A}(\mathbf{t}^*, \mathbf{a}^*) \cdot \mathbf{r}^* = (1, 0, \dots, 0)^T, \\
& \mathbf{A}(\mathbf{t}^*, \mathbf{a}^*) \cdot \begin{pmatrix} h^* \\ \lambda^* \end{pmatrix} = \mathbf{c}(\mathbf{t}^*, \mathbf{a}^*), \\
& t_j^* \in E(\lambda^*) \text{ and } \alpha_j^* = \arg(\epsilon(t_j^*; \lambda_j^*)), j = 1, 2, \dots, 2n + 1.
\end{aligned}$$

This can be viewed as the complex extension of the **second algorithm of Remez** for real Chebyshev approximation. It should be noted that the **Haar condition** guarantees the existence of  $n + 1$  extremal points with positive weights in the above expression. In the complex case, it is possible that fewer than  $2n + 1$  points exist or that extremal points have a zero weight. Note that this approach leads to a low-dimensional optimization problem and does not depend on the number of extremal points. Moreover, upper and lower bounds for the minimal deviation are available, at no extra cost, at any step of the algorithm. The linear semi-infinite optimization problem is solved by means of the Simplex algorithm. In a second phase, the discrete solution is improved by solving a nonlinear system of equations via Newton's method. Note that the convergence of Newton's method depends on the correct number of extremal points and on a good starting point. In the case of the inverse gamma function, this second phase was not used.

**Results**

We found that

$$\|f - p\|_\infty = \mathbf{0.2143352345}.$$

We include as a plot some of the output generated by COCA in Figure 2, as well as a plot showing the reciprocal gamma function on the unit disk, its cubic approximant, and the error in absolute value between the two in Figure 3.

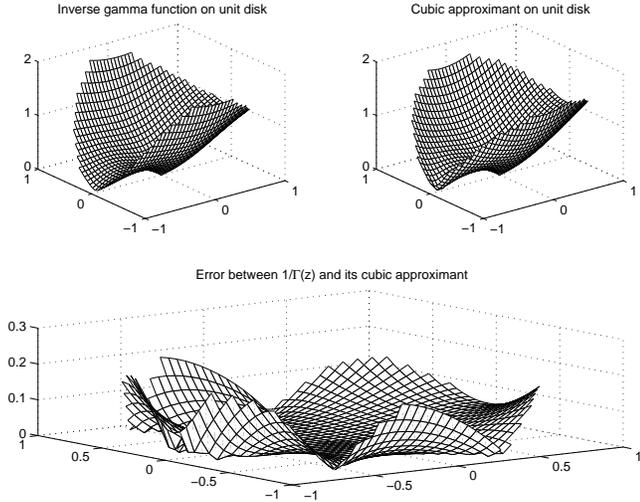


Figure 3: Plots from Problem 5

## References

- [1] Bernd Fischer and Jan Modersitzki, *An algorithm for complex linear approximation based on semi-infinite programming*, Numerical Algorithms 5, 287-297, 1993.
- [2] Einar Hille, *Analytic Function Theory, Vol I*, Chelsea, New York, 1962.
- [3] Lloyd N. Trefethen, *Spectral Methods in Matlab*, SIAM, Philadelphia, 2000.

## Solution Method 2 by Hoang Nguyen

We did a direct search for the coefficients of the cubic polynomial that minimizes the supremum norm. The Matlab routine was `fminsearch` (Nelder–Mead method). The unit disk was discretized in polar coordinates, and the inverse gamma function was approximated using Taylor series to 50 terms. This method yielded an answer that agrees to 9 digits with our other method that used Chebyshev polynomials.

## Problem 6

### Solution Method 1 by Christiaan Stolk

Let  $A_n$  be the event that the flea returns to the origin after  $2n$  steps for the first time after it left the origin at time 0. (Note that it will never be at the origin after an odd number of steps.) The total probability of the flea coming back to the origin is given by

$$\sum_{n=1}^{\infty} P(A_n). \tag{2}$$

To derive an expression for  $P(A_n)$  we first need some notation. Let the random variable  $X_n$  denote the east-west position after  $n$  steps, and  $Y_n$  the north-south position of the flea after  $n$  steps. Let  $K_{k,n}$  be the event that after  $2n$  steps, the flea has taken  $2k$  steps in the east-west direction.

The distribution of  $X_{2n}$ , given there have been  $2k$  steps in the east-west direction, is binomial, and the probability for  $X_{2n} = 0$  is given by

$$P(X_{2n} = 0 | K_{k,n}) = \left(\frac{1}{2} - 2\epsilon\right)^k \left(\frac{1}{2} + 2\epsilon\right)^k \binom{2k}{k}.$$

Similarly the probability that  $Y_{2n} = 0$ , given that there have been  $2k$  steps in the east-west direction and hence  $2(n - k)$  steps in the north-south direction, is given by

$$P(Y_{2n} = 0 | K_{k,n}) = \left(\frac{1}{2}\right)^{2(n-k)} \binom{2(n-k)}{n-k}.$$

The probability that  $K_{k,n}$  occurs also has a binomial distribution

$$P(K_{k,n}) = \left(\frac{1}{2}\right)^{2n} \binom{2n}{2k}.$$

Let  $B_n$  be the event that after  $2n$  steps the flea is at  $(0, 0)$ . If the number of steps in the east-west direction is equal to  $2k$ , then this chance is simply the product  $P(X_{2n} = 0 | K_{k,n}) P(Y_{2n} = 0 | K_{k,n})$ ,

which is calculable from the expressions above. The total probability that  $B_n$  occurs is obtained by adding such contributions for different  $k$ , multiplied by the probability that each of them occurs:

$$\begin{aligned} P(B_n) &= \sum_{k=0}^n P(B_n|K_{k,n})P(K_{k,n}) \\ &= \sum_{k=0}^n P(X_{2n} = 0|K_{k,n})P(Y_{2n} = 0|K_{k,n})P(K_{k,n}). \end{aligned}$$

This can easily be evaluated given the formulas above.

To get  $P(A_n)$  from  $P(B_n)$ , the probability that the flea has been at the origin before must be subtracted:

$$P(A_n) = P(B_n) - \sum_{k=1}^{n-1} P(A_k|B_n)P(B_n) = P(B_n) - \sum_{k=1}^{n-1} P(B_n|A_k)P(A_k).$$

It is not difficult to calculate the probability that the flea is at the origin after  $2n$  steps, given that it was at the origin after  $2k$  steps, this is simply given by  $P(B_{n-k})$ . Thus we obtain

$$P(A_n) = P(B_n) - \sum_{k=1}^{n-1} P(B_{n-k})P(A_k)$$

The probabilities  $P(A_n)$  can now be calculated in sequence.

The expected position of the flea after  $n$  steps is given by  $(2n\epsilon, 0)$ . It follows by the law of large numbers that the probability of the flea returning to zero becomes exponentially small for nonzero  $\epsilon$  and large time. Therefore the sum (2) has good convergence properties. The value of  $\epsilon$  where (2) is equal to  $1/2$  can now be calculated by a rootfinding algorithm. Using high precision calculus in Mathematica<sup>TM</sup>, this gives the answer 0.0619139544739909428481752164732, and with a little more computing time additional digits can easily be computed.

## Solution Method 2 by Daniel Reynolds and Chris Husband

We may describe our problem as a Markov chain. For this, we need to build what is called a transition matrix, and give it an initial state. The general idea is as follows.

**Grid and Initial State:** In order to be able to calculate anything, we must first reduce our working grid so that it is no longer infinite. This will be a source of error, since by disregarding possible points far from the origin, we lose some probability. To this end, we may calculate a few distinct probabilities on whether the flea would make it to a certain point out and then back to the origin. For instance, the probability that the flea will reach a position 40 to the north or south and then make it back to the origin is much less than  $10^{-20}$ , so to be safe (since we must sum the probabilities from many points) we extend the working grid 50 in those directions. We must further consider that for  $\epsilon > 0$ , the probability of heading east and returning to the origin is potentially much higher, so we extend the grid 50 to the west and 100 to the right.

We must then think of a clever means of indexing each integer lattice point on this grid, so that we have for instance

$$1 \rightarrow (0, 0), \quad 2 \rightarrow (0, 1), \quad 3 \rightarrow (1, 0), \quad 4 \rightarrow (0, -1), \quad \dots$$

Under a labeling such as this, our initial state may then be labeled as

$$X^0 = [1, 0, 0, 0, \dots]^T,$$

meaning that the flea will be at the origin at the initial time with probability 1. This labeling is actually reordered to yield a transition matrix that is banded with 9 diagonals.

**Transition Matrix:** For the transition matrix, we now write the entries as  $P_\epsilon(i, j) =$  the probability that from position  $X(j)$ , the flea will be in position  $X(i)$  after two jumps. We use two jumps since it will take a multiple of two jumps for the flea to return to any spot. We also define the augmented transition matrix  $\bar{P}_\epsilon$  as  $P_\epsilon$  with the diagonal entry corresponding to the origin set to one, and the rest of the column set to zero. This matrix will be used in order to ‘trap’ the flea at the origin once it has landed there.

**Probability Function:** Once we have the transition matrices  $P_\epsilon$  and  $\bar{P}_\epsilon$  and initial state  $X^0$ , we may then find the probability of the flea reaching any spot  $j$  at step  $2n$  by first obtaining  $X^2 = P_\epsilon * X^0$ , and then

$$P[\text{flea in position } j \text{ at step } 2n] = (\bar{P}_\epsilon^{n-1} X^2)(j).$$

Thus if we want to find the probability that the flea reaches the origin again at the  $n^{\text{th}}$  time step, we merely calculate the entry  $X^n$  corresponding to the index of the origin. (This will converge as  $n \rightarrow \infty$ .) This is because in the calculation of  $X^2$  we calculate the exact probability that the flea returns to the origin after the first two steps. Then by using the augmented transition matrix  $\bar{P}_\epsilon$  for the remaining iterations, we essentially sum up the probabilities that the flea will get back to the origin and stay there from then on. Furthermore, we can do each of these multiplications very quickly using sparse matrix multiplication.

**Optimization:** Standard optimization routines such as Newton’s method are infeasible for finding the desired  $\epsilon$ , since derivatives of the above routine are difficult. A more rudimentary bisection method works well. We know that  $\epsilon$  must be contained within the interval  $(0, 1/4)$ , and that  $P_0 > \frac{1}{2}$  and  $P_{1/4} < \frac{1}{2}$ . We also know that the probability uniformly decreases as  $\epsilon$  increases. Thus on the first iteration we try  $\epsilon = 1/8$ . Depending on the probability  $P_{1/8}$ , we will then move to either  $1/16$  or  $3/16$ . In this manner we may narrow down the value of  $\epsilon$  by a power of 2 at each iteration. Thus after at least 32 overall iterations, the value of  $\epsilon$  should be accurate to at least 10 digits of accuracy.

**Results:** The solution obtained using this method was 0.06191395447398. This computation did not take long, however, so more digits could be found relatively simply until the numerical roundoff error of floating point arithmetic takes control.

### Solution Method 3 by Hoang Nguyen

The following code computes the probability of the flea having returned to the origin sometimes within a number of steps `nt`, for a given bias parameter `e`. First, we use the multinomial distribution to compute the probability that the flea is at the origin after exactly `n` steps (`n` is even, from 2 to `nt`). For each possible step combination  $(i_N, i_S, i_E, i_W)$ , this is given by

$$P = \frac{(i_N + i_S + i_E + i_W)!}{i_N! i_S! i_E! i_W!} (p_N)^{i_N} (p_S)^{i_S} (p_E)^{i_E} (p_W)^{i_W},$$

where  $i_N$  is the number of North steps and  $p_N$  is the probability of going North in one step, etc. To be at the origin after `n` steps would require the constraints  $i_N = i_S, i_E = i_W$  and  $i_N + i_S + i_E + i_W = n$ . These probabilities were summed up over all feasible step combinations for a given `n`. From these probabilities, we next compute the probability that the flea has returned to the origin for the first time after exactly `n` steps. We then sum up these ‘first-time’ probabilities to give the desired probability `p`. It is a simple search to find the bias parameter that gives 0.5 for `p`. We found 0.06191395447399 for this bias parameter, using 3000 time steps. If more digits are desired then we can use the variable precision arithmetic feature in Matlab.

```
function p = f6(e,nt)
```

```

% F6      Function to compute the probability of returning to the origin
%         sometimes during a biased random walk.
%
%   p = f6(e,nt)
%
% INPUT:
%   e     the bias parameter
%   nt    the maximum number of time steps (optional, even, defaults to 2000)
%
% OUTPUT:
%   p     probability of having returned to origin within nt time steps

if (nargin < 2 | nt > 100000 | nt < 2 | mod(nt,2) > 0)
    nt = 2000;           % default maximum number of steps to compute
end
pN = 0.25;              % probability of jumping North
pS = 0.25;              % probability of jumping South
pE = 0.25 + e;         % probability of jumping East
pW = 0.25 - e;         % probability of jumping West
logp = log([pN pS pE pW]); % natural log of probability
dprobn = zeros(nt/2,1); % probability that, after exactly n steps,
                        % the flea is currently at the origin
for n = 2:2:nt         % for each even number of steps
    nhalf = n/2;       % maximum allowed steps in one direction
    probn = 0;         % cumulative probability of being at 0 for this n
    for j = 0:nhalf    % consider all feasible step combinations
        iN = j;        % number of North steps
        iS = j;        % number of South steps
        iE = nhalf - j; % number of East steps
        iW = iE;       % number of West steps
        logpdir = [iN iS iE iW] .* logp; % log([(pN^iN), (pS^iS), (pE^iE), (pW^iW)])
        % compute coefficient of multinomial distribution using logarithms
        num = [1:n];   % numerator of coef
        den = [(1:iN) (1:iS) (1:iE) (1:iW)]; % denominator of coef
        logratio = log(num./den); % avoid overflow by summing logarithms
        logsum = sum(logpdir) + sum(logratio); % sum of logs = log of products
        pj = exp(logsum); % probability for this step combo
        probn = probn + pj; % sum over all possible step combo for this n
    end
    dprobn(nhalf) = probn; % save cumulative probability for this n
    if (mod(n,100) == 0)
        fprintf('n = %d,   probn = %22.16e\n',n,probn); % display progress
    end
end

pfirst = zeros(nt/2,1); % probability of returning to 0 for the first time
for i = 1:nt/2
    pfirst(i) = dprobn(i);
    for j = 1:i-1
        pfirst(i) = pfirst(i) - dprobn(j)*pfirst(i-j);
    end
end
end
ppos = find(pfirst>0); % skip negative probabilities due to rounding
p = sum(pfirst(ppos)); % probability of returning by this step

```

## Problem 7

### Solution Method 1 by Hoang Nguyen

We built the matrix  $A$  according to specifications, storing it in a sparse format. Using this matrix we then solved the system  $Ax = e_1$ , where  $e_1$  is the first column of the identity matrix having the

same dimension as  $A$ . The first element of  $x$  is the  $(1, 1)$  element of  $A^{-1}$ . We first used the backslash ( $\backslash$ ) command in Matlab to solve the system and obtained  $7.25078346268401e - 01$ . We also used the conjugate gradient method for verification and got the same answer to 15 digits. To check the reliability of our answer, we computed the residual vector and estimated the condition number of the  $A$  matrix. All evidence indicated that the answer is reliable. We also computed the answer for smaller matrices with the same structure to check the stability of our answer. The sequence of answers did approach the answer for size  $n = 20000$ . The following code computes the answer for a given matrix order, using both sparse direct method and CG.

```
function t = f7cg(N)
P = primes(224737)';           % list of first 20000 prime numbers
n = min(N,length(P));        % order of matrix to use
e = ones(n,1);               % off-diagonals elements
m = ceil(log2(n));           % number of sub-diagonals
A = spdiags(P(1:n),0,n,n);    % put primes on diagonal
for i=0:m                     % build matrix according to specs
    j = 2^i;
    if (j < n)
        A = spdiags(e, j, A);    % put in super-diagonal elements
        A = spdiags(e, -j, A);   % put in sub-diagonal elements
    end
end
e1 = sparse([1],[1],1,n,1,1); % right hand side of linear system
% direct method
x_direct = A \ e1;           % sparse direct method
invA11_direct = x_direct(1)  % (1,1) element of inv(A), direct method
% Conjugate Gradient
tol = 1e-16;                 % stopping tolerance
maxit = 2*N;                 % maximum number of iterations
x0 = zeros(n,1);            % initial guess
[x,flag,relres,iter,resvec] = pcg(A,e1,tol,maxit,[],[],x0);
invA11_cg = x(1)             % (1,1) element of inv(A), CG method
flag                       % exit flag
iter                       % number of iterations used
relres                     % relative residual
% check residual
res = A * x - e1;           % residual
resNorm1 = norm(res,1)      % 1-norm of residual
resNorm2 = norm(res,2)      % 2-norm of residual
resNormInf = norm(res,Inf)  % Inf-norm of residual
res_1_10 = res(1:10)        % first 10 elements of residual
condA = condest(A)          % condition number estimate
```

## Solution Method 2 by Christiaan Stolk

It was already remarked in the previous approach to this problem that we must solve the system  $Ax = e_1$ , where  $e_1$  is the vector with first element equal to 1 and all other elements equal to 0. The first element of  $x$  is the answer to the problem. Let  $D$  be the diagonal part of  $A$ , and let  $B = A - D$  be the off-diagonal part. The inverse  $A^{-1}$  satisfies

$$A^{-1} = (D + B)^{-1} = (D(1 + D^{-1}B))^{-1} = (1 + D^{-1}B)^{-1}D^{-1}.$$

If the (operator) norm of  $D^{-1}B$  is less than 1, then

$$(1 + D^{-1}B)^{-1} = \sum_{j=0}^{\infty} (-D^{-1}B)^j.$$

In that case  $x$  is given by the following expression

$$x = \sum_{j=0}^{\infty} (D^{-1}B)^j D^{-1}e_1. \quad (3)$$

We computed an approximation to  $x$  by taking a finite, but large sum to approximate (3). We did not prove that the norm of  $D^{-1}B$  is smaller than 1, but the vectors  $(D^{-1}B)^j D^{-1}e_1$  became exponentially small when  $j$  became large (with length  $\approx 10^{-132}$  after 2000 steps). This is an indication that  $D^{-1}B$  has norm  $< 1$ . For the first component of  $x$  we found 0.725078346268401167468687719251.

## Problem 8

### Solution Method 1 by Christiaan Stolk

We define the domain  $\Omega = [0, 2] \times [-1, 1]$ , and assume that the special side with temperature 5 is the one given by  $x_1 = 0$ . We transform this problem to a problem on the whole plane in a few steps. Let  $u$  be a solution to the original problem. We define  $u_1(x, t)$  for  $x$  in  $[-2, 2] \times [-1, 1]$  by

$$u_1(x, t) = \begin{cases} u(x, t) & \text{if } x_1 > 0 \\ 10 - u(x, t) & \text{if } x_1 < 0. \end{cases}$$

Now  $u_1$  satisfies a heat equation on  $\Omega_1 = [-2, 2] \times [-1, 1]$  with initial value equal to  $u_1(x, 0) = 10 - 10H(x_1)$ , where  $H(z)$  is the Heaviside function given by  $H(z) = 1$ , for  $z > 0$ , and by  $H(z) = 0$  for  $z < 0$ . The boundary values are also equal to  $10 - 10H(x_1)$  on  $\partial\Omega_1$ .

This problem can be transformed to a problem with 0 boundary data by defining

$$u_2(x, t) = u_1(x, t) - 10 + 10H(x_1).$$

The function  $u_2$  solves a heat equation with a source

$$\partial_t u_2 = \Delta u_2 - 10\delta'(x_1),$$

on  $\Omega_1$ , satisfies homogeneous Dirichlet boundary conditions on  $\partial\Omega_1$ , and has zero initial conditions.

Because we have zero Dirichlet conditions on a rectangle, we can do an odd periodic extension. Then  $u_2$  is given by the solution of an equation on the whole plane  $\mathbb{R}^2$ . Denote by  $I_{[a,b]}(z)$  the indicator function of an interval that is 1 inside the interval, and 0 outside the interval. The source for the heat equation on the whole plane for  $u_2$  is given by the following distribution,

$$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (-1)^{j+1} 10\delta'(x_1 - 4i) I_{[2j-1, 2j+1]}(x_2).$$

The Green's function for a heat equation on the whole plane is given by

$$\frac{1}{4\pi t} e^{-\frac{x_1^2 + x_2^2}{4t}}.$$

We thus find the following expression for the solution  $u_2$ , and hence for  $u$

$$u_2(x, t) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (-1)^j \int_{2j-1}^{2j+1} dz \int_0^t ds \frac{5(x_1 - 4i)}{4\pi(t-s)^2} e^{-\frac{(x_1 - 4i)^2 + (x_2 - z)^2}{4(t-s)}}.$$

Because the exponential decreases rapidly as the spatial variables increase, only a few terms of the summation contribute in the range of times that is of interest. The time integration can be done symbolically, so only a small number of 1D space integrations remains. The time when the  $u(1, 0) = 1$  can now be computed using a root finding algorithm (such as FindRoot in Mathematica<sup>TM</sup>). This gives the result 0.424011387033688363797433668593 (more digits can easily be computed).

## Solution Method 2 by Daniel Reynolds

We may use Fourier series to solve the heat equation. For Dirichlet boundary conditions on this domain and the operator  $-\Delta(\cdot)$ , we have the eigenpairs

$$\left\{ \sin\left(\frac{n\pi x}{2}\right) \sin\left(\frac{m\pi y}{2}\right), \frac{n^2\pi^2}{4} + \frac{m^2\pi^2}{4} \right\}.$$

If we then construct a function  $p(x, y)$  that satisfies the inhomogeneous Dirichlet boundary conditions (even only pointwise), we define  $v(x, y, t) = u(x, y, t) - p(x, y, t)$  where  $u(x, y, t)$  is the solution to the original heat equation. Then the original differential equation may be written as

$$\begin{aligned} v_t(x, y, t) - \Delta v(x, y, t) &= \Delta p(x, y) \\ v(x, y, 0) &= -p(x, y) \\ v(x, y, t)|_{\text{boundary}} &= 0 \end{aligned} \tag{4}$$

We may then expand the functions  $v(x, y, t)$ ,  $\Delta p(x, y)$ , and  $-p(x, y)$  along the Fourier coefficients  $a_{mn}(t)$ ,  $b_{mn}$ , and  $c_{mn}$ , respectively. We may solve for the coefficients  $a_{mn}(t)$  according to the linear ODE

$$\begin{aligned} \frac{d}{dt} a_{mn}(t) + \left( \frac{n^2\pi^2}{4} + \frac{m^2\pi^2}{4} \right) a_{mn}(t) &= b_{mn} \\ a_{mn}(0) &= c_{mn}. \end{aligned} \tag{5}$$

This may be solved analytically in time for the coefficients  $a_{mn}(t)$ .

Furthermore, the coefficients  $b_{mn}$  result from the calculation

$$b_{mn} = \frac{1}{4} \int_{-1}^1 \int_{-1}^1 \Delta p(x, y) \sin\left(\frac{n\pi x}{2}\right) \sin\left(\frac{m\pi y}{2}\right) dx dy. \tag{6}$$

Although we have only a set of point values for  $p(x, y)$ , we may use Green's theorem (integration by parts) to evaluate the integral. Using this, we find that the coefficients  $b_{mn}$  may be written entirely in terms of the coefficients  $c_{mn}$ .

To improve computational efficiency, note that the temperature at the center of the plate will only depend on the odd Fourier coefficients, since the even ones correspond to basis functions having zero value at the center. Lastly, after examining the construction of  $a_{mn}$  from the coefficients  $b_{mn}$  and  $c_{mn}$ , we see that we may write the temperature at the center of the plate at time  $t$  as the double sum

$$u(t) = \sum_{m=1,3,\dots}^{N_x-1} \sum_{n=1,3,\dots}^{N_y-1} \left( \frac{40m}{n\pi^2(m^2 + n^2)} \right) \left( 1 - \exp\left(\frac{-t\pi^2(m^2 + n^2)}{4}\right) \right) (-1)^{\frac{m+3}{2}} (-1)^{\frac{n+3}{2}}.$$

This is then put into a Newton's method to determine the exact time that the temperature at the center of the plate reaches 1. The major source of error from this method is the fact that  $N_x$ ,  $N_y$  must be very large in order to sufficiently approximate the boundary function  $p(x, y)$ . Using this method, along with the summation scheme discussed in problem 10, the time that the center of the plate reaches value 1 was calculated to be  $t = 0.42401138703368824$ .

## Solution Method 3 by Hoang Nguyen

We used the Matlab PDE Toolbox as a quick method to find a few digits for this problem and as a check on the two more accurate methods. It was clear that a standard finite element method would not be able to yield 10 digits of accuracy on our hardware. We used the Matlab function `poimesh` to generate a regular mesh, then used `parabolic` to solve the heat equation. For triangular elements of size 0.01 ( $nx = 200$ ) and time increment of 0.0001, we obtained an answer of 0.4240.

## Problem 9

### Solution Method 1 by Christiaan Stolk

The main problem here is the accurate numerical evaluation of the integral  $I(\alpha)$ , because the integrand is oscillatory near  $x = 2$ . To solve this we do a transformation of variables

$$y = \frac{1}{2 - x}.$$

The integral becomes

$$I(\alpha) = \int_{1/2}^{\infty} f(y)g(y) \, dy,$$

where

$$f(y) = (2 + \sin(10\alpha)) \frac{1}{y^2} \left(2 - \frac{1}{y}\right)^\alpha,$$
$$g(y) = \sin(\alpha y).$$

We choose a number  $a$  between  $1/2$  and  $\infty$ , and write

$$I(\alpha) = \int_{1/2}^a f(y)g(y) \, dy + \int_a^{\infty} f(y)g(y) \, dy.$$

The first term can be evaluated by straightforward numerical evaluation (such as `NIntegrate` in Mathematica<sup>TM</sup>). To approximate the second part, let us call it  $I_2(\alpha)$ , we do repeated partial integration. After a single partial integration we have

$$I_2(\alpha) = - \int_a^{\infty} f'(y)g^{(-1)}(y) \, dy + f(y)g^{(-1)}(y) \Big|_a^{\infty}.$$

Here we denote by  $f^{(k)}$  the  $k^{\text{th}}$  derivative of  $f$ , for  $k$  positive, and the  $k^{\text{th}}$  primitive of  $f$  for  $k$  negative. After  $k$  repeated partial integrations we find

$$I_2(\alpha) = (-1)^k \int_a^{\infty} f^{(k)} g^{(-k)}(y) \, dy + \sum_{j=0}^{k-1} (-1)^j f^{(j)}(y) g^{(-j-1)}(y) \Big|_a^{\infty}. \quad (7)$$

The primitives  $g^{(-j)}$  can easily be calculated explicitly. The derivatives  $f^{(j)}$  can be computed automatically using mathematical software. When  $a$  and  $k$  are sufficiently large, then the derivatives  $f^{(k)}(y)$ ,  $y \geq a$  become very small. In addition the boundary terms at infinity in (7) are 0. For sufficiently large  $a$  and  $k$  we therefore find the following approximation for  $I(\alpha)$

$$I(\alpha) \approx \int_{1/2}^a f(y)g(y) \, dy - \sum_{j=0}^{k-1} (-1)^j f^{(j)}(a)g^{(-j-1)}(a). \quad (8)$$

The next step is to find the maximum of  $I(\alpha)$  and the corresponding  $\alpha$ . A plot of an approximation of  $I(\alpha)$  (by taking the integral from  $x = 0$  to  $x = 1.999$ ) is given in Figure 4. The maximum is assumed around  $\alpha = 0.8$ . Using the command `FindMinimum` in Mathematica<sup>TM</sup> the maximum of  $I(\alpha)$  and the corresponding value of  $\alpha$  are established to high accuracy. This yields 0.7859336743503714545652 for the answer.

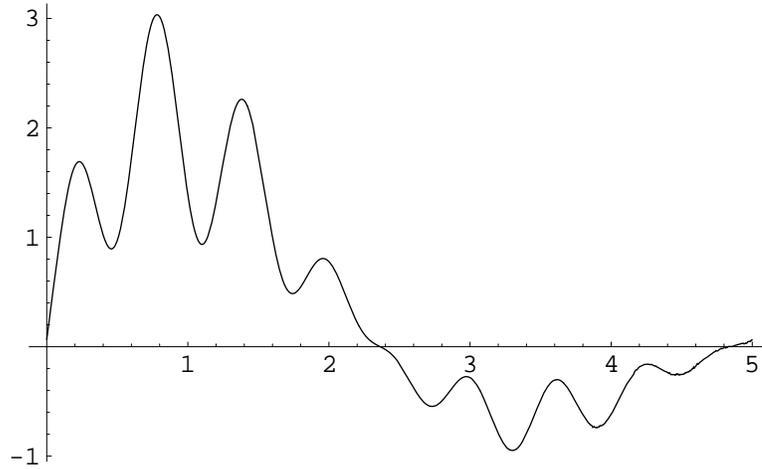


Figure 4: Plot of an approximation to  $I(\alpha)$  as a function of  $\alpha$  in problem 9

## Solution Method 2 by Hoang Nguyen

We used the transformation  $z = 2 - x$  to turn the objective function into

$$I(\alpha) = (2 + \sin(10\alpha)) \int_0^2 (2 - z)^\alpha \sin(\alpha/z) dz.$$

The integral was computed using Maple, which was able to handle the singularity at  $z = 0$ . The search for the optimal  $\alpha$  was done using a *for / do* loop within Maple, gaining one digit of precision for each loop. At first we used only 20 digits of precision to narrow down the search range (Maple was able to do this quickly.) Near the optimal value we increased the precision to 48 digits. The optimal  $\alpha$  to 22 digits is

$$0.7859336743503714545652,$$

giving an objective function value of

$$3.0337325864854936325378726835128253682157215.$$

## Problem 10

### Solution Method 1 by Christiaan Stolk

Define the domain

$$\Omega = [-5, 5] \times [-1/2, 1/2].$$

Let  $u(x, t)$  be a probability density function for the particle. Assuming that the particle is absorbed at the boundary, this probability density function satisfies a heat equation on  $\Omega$  with Dirichlet boundary condition, and initial value equal to a  $\delta$  function

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u && \text{on } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \\ u(x, 0) &= \delta(x). \end{aligned}$$

Because the domain is rectangular, we can do an *odd periodic extension*, of  $u$  in  $x$  to the whole of  $\mathbb{R}^2$ . The solution is equal to the solution of a heat equation on the whole plane, with initial value equal to the odd periodic extension of  $\delta(x)$

$$g(x) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (-1)^{i+j} \delta(x - (10i, j)).$$

The probability of the particle crossing a certain line interval is given by the time integral of the particle flux across the line interval. This probability for a single  $\delta$  source in the plane  $\mathbb{R}^2$  is simply given by  $\frac{\alpha(i, j)}{2\pi}$ , where  $\alpha(i, j)$  is the opening angle from which the line interval is seen from the source point. This is due to the circular symmetry that is present for a single source in the plane. For the line interval from  $(-5, -1/2)$  to  $(-5, 1/2)$ , and a source at  $(10i, j)$ ,  $i \geq 0$  this is given by

$$\frac{1}{2\pi} \left( \arctan \left( \frac{1/2 + j}{5 + 10i} \right) - \arctan \left( \frac{-1/2 + j}{5 + 10i} \right) \right).$$

For the total flux we obtain

$$\phi = 2 \cdot 2 \cdot \frac{1}{2\pi} \sum_{i=0}^{\infty} \sum_{j=-\infty}^{\infty} (-1)^{i+j} \left( \arctan \left( \frac{1/2 + j}{5 + 10i} \right) - \arctan \left( \frac{-1/2 + j}{5 + 10i} \right) \right).$$

There is one factor 2 because the contributions from sources  $i \leq -1$  which are equal to the contributions for  $i \geq 0$ . There is another factor 2 because there are two short sides of the rectangle that give the same contribution.

We rewrite the sum over  $j$  as follows:

$$\begin{aligned} \phi &= 2 \cdot 2 \cdot 2 \cdot \frac{1}{2\pi} \sum_{i=0}^{\infty} (-1)^i \\ &\quad \times \left( \arctan \left( \frac{1/2}{5 + 10i} \right) + \sum_{j=1}^{\infty} (-1)^j \left( \arctan \left( \frac{1/2 + j}{5 + 10i} \right) - \arctan \left( \frac{-1/2 + j}{5 + 10i} \right) \right) \right). \end{aligned}$$

Now each of the contributions occurs twice, with the same sign (note that  $1/2 + j = -1/2 + (j+1)$ ). So this sum can be rewritten as

$$\begin{aligned} \phi &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot \frac{1}{2\pi} \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} (-1)^{i+j} \arctan \left( \frac{1/2 + j}{5 + 10i} \right) \\ &= \frac{8}{\pi} \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} (-1)^{i+j} \arctan \left( \frac{1/2 + j}{5 + 10i} \right). \end{aligned} \tag{9}$$

Expression (9) is an alternating sum in two dimensions. When a straightforward summation is done the convergence is poor. We perform an additional trick to obtain better convergence. We describe this here for a one dimensional alternating sum

$$\sum_{j=-\infty}^{\infty} (-1)^j f(j). \tag{10}$$

We assume that the higher order derivatives  $\frac{d^k f}{dx^k}(x)$  of  $f(x)$  go to zero increasingly fast when  $k$  increases.

Let

$$w_{i,k} = \frac{1}{2^{k-1}} (-1)^i \binom{k}{i}.$$

We define  $g_k(i) = \sum_{j=0}^k w_{j,k} f(i+j)$ ,  $k \geq 1$ . Observe that  $g_k(i)$  is equal to a finite difference approximation of  $(-1)^k 2^{-k+1} \frac{d^k f}{dx^k}(i+k/2)$ , and that it therefore goes to zero increasingly fast when  $k$  increases. Expression (10) is equal to

$$\sum_{j=-\infty}^{\infty} g_k(2j).$$

When  $k$  is large this sum converges fast, and can be used for numerical evaluation of (10) with high accuracy. This yields the answer  $3.83758797925122610340713318620 \times 10^{-7}$ ; more digits are not hard to compute.

## Solution Method 2 by Chris Husband

At first, the magnitude of the solution to this problem appeared to be extreme and some of us doubted the accuracy of the result. While obviously a simulation of the particle could never measure this probability because of the convergence of  $1/\sqrt{n}$  (it would take  $10^{14}$  independent simulations to get the first digit), a simulation could be applied to at least validate the first method by applying simpler and more naive simulations to smaller problems that could be solved sufficiently accurately with regards to the  $1/\sqrt{n}$  convergence rate. This method considered plates of dimension  $1 \times 2$  and  $1 \times 3$  and a particle that chose a direction randomly (between 0 and  $2\pi$ ). The particle then took a small step in that direction. The particle continued to take these steps until it went past a boundary. The simulation ran multiple walks, tabulating which boundary the particle crossed and averaging the results to obtain an estimated probability. While this method suffers serious drawbacks, such as slow convergence rates and finite step size, it did help provide validation of the other method. In both problems, with step sizes of .01 and .005 using 10,000 iterations, we could roughly expect 2 digits of accuracy, and in both cases the results match those of the more thorough method. Further refinement of the problem with a million iterations gave us more accuracy. Some of these results encouraging the solution from method 1 are given in Table 2.

Domain	Step Length	Probability	Method 1
$1 \times 2$	0.01	0.1119	0.109769799
	0.005	0.1098	
$1 \times 3$	0.01	0.0249	0.0228733
	0.005	0.0225	

Table 2: Brownian motion on two smaller domains with varying step size, all using 10,000 iterations.