Solving for the Steady-State Solution of the Heat Equation in a Cubic Satellite

Jonathan Bunton

November 30, 2017

Abstract

Before launching objects into low orbit space, it is prevalent to study their expected behavior in given conditions. One particular behavior worth investigating is the eventual thermal characteristics of the system. To do this, we can utilize the fact that in general, the temperature of a material over time $T(x, y, z, t)$ is governed by the heat equation. [2] This partial differential equation is given by:

$$\nabla^2 T = \frac{1}{\alpha}\frac{dT}{dt}$$

In the eventual steady-state situation, the temperature $T(x, y, z, t)$ ceases to change with time. Put mathematically, we seek a solution to:

$$\nabla^2 T = 0.$$

To even begin to solve this equation, we require proper boundary conditions. In this particular case, we consider a tidally locked 0.1 meter cubic mass of polystyrene foam, coated with polysilicon and containing a 0.4 meter cubic aluminum mass which stays at a constant 100 K. This will serve as our "satellite." We assume one side remains facing the sun and absorbs thermal energy holding it at (governed by the Stefan-Boltzmann law) and the face opposite the sun remains at the temperature of the cosmic microwave background. [5, 4] In addition, the faces between these two assume a linear gradient between the two temperatures which is logical for the long-term behavior. The only region remaining with an unknown steady-state temperature distribution is the polystyrene foam in the center.

To solve for this distribution, divide the box into a 3D mesh and use the method of finite differences to approximate the second derivative at each point. We then solve the resulting system of equations for all internal points alongside the boundary condition using numerical methods, eventually yielding a numerical $T(x, y, z)$ that indicates the steady-state temperature at any point on the satellite.

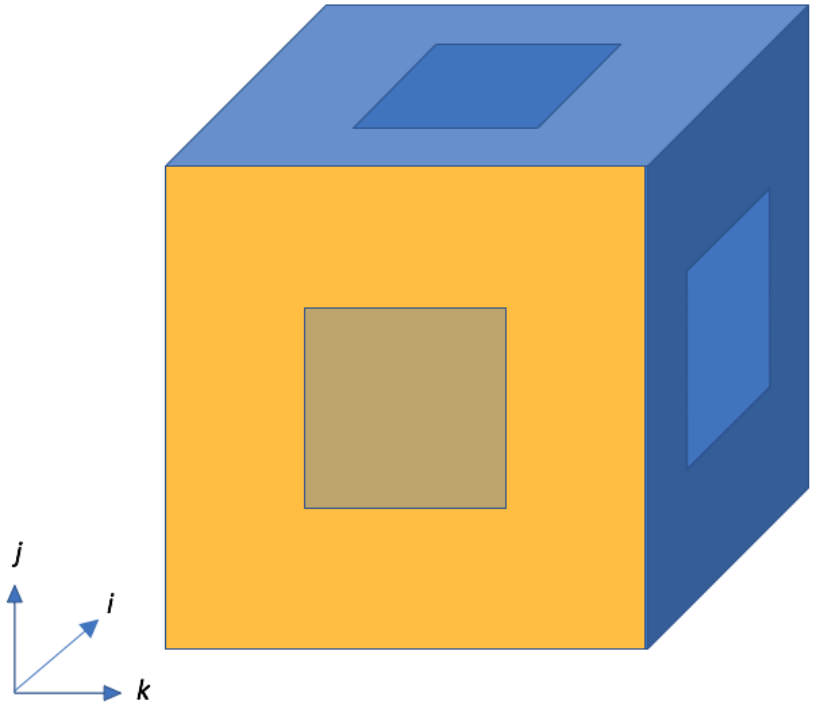


Figure 1: An image indicating the orientation of our cubic satellite in space, alongside the position of the interior cube, marked on the outer faces.

This system of equations can be solved a variety of ways, however, this paper analyzes two possible methods–computing a LU factorization and solving, and utilizing pre-written subroutines within $LAPACK$. [3, 1]

The results indicate a linear trend away from the satellite face closest to the sun, with exponential curves downward to the internal aluminum, as it is generally cooler than the surrounding temperature. The hand-written LU factorization method takes noticeably longer than $LAPACK$, as is expected. Though there is marginal difference, the variation between the solutions produced by both methods coincide within $10^{-12}\%$.

Results

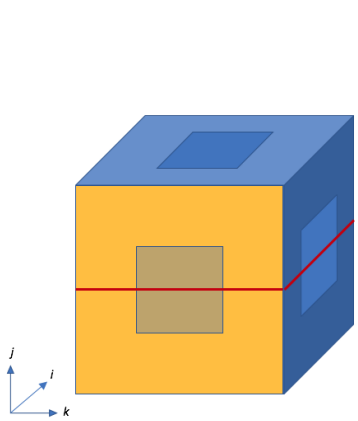
Our satellite is oriented with the axes i, jk as shown in figure 1. The face marked with yellow, $i = 1$, is facing the sun. This orientation is consistent through all plots.

Temperature Results

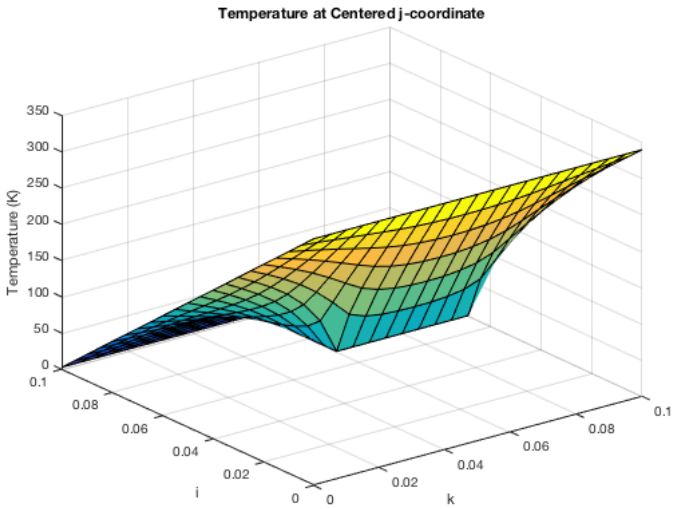
Because our result data is a temperature at each point in 3D space, we are left with four dimensional results. Visualization of this is easiest with 3D cross-sectioned images, which are shown below.

If we first look at cross-sections of the temperature with constant j and k in figures 2b and 3b, we see exactly what we would expect on the edges: linear trends on the boundaries. This is how we initialized our satellite faces through i, so this makes sense. In the center of the box, we have a square with constant temperature 100 K–the aluminum center. Between these two boundaries, the linear gradient through i persists, interrupted only by a decaying curve between the gradient and the aluminum center's 100 K.

The decaying curve makes sense, since the aluminum center at 100 K is generally cooler than the outside faces, so it acts almost as a heat sink, cooling the immediate area surrounding it. The linear trend throughout the system is consistent with the steady state solution to the heat equation as well, which is linear. [2]



(a)

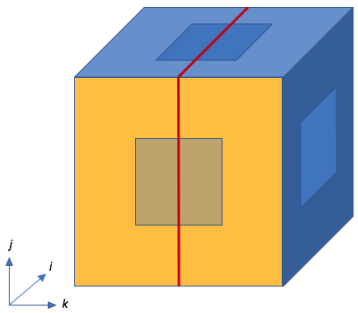


(b)

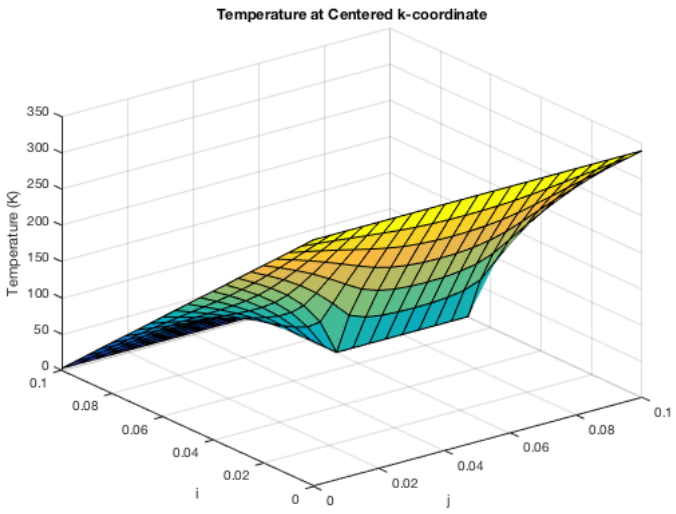
Figure 2: A plot of temperature through a cross-section at the center of the j-axis. There is a noticeable exponential towards the aluminum center temperature (100 K), however, the trend through i is linear.

Our cross section in k shown in fig. 3b is an identical plot to fig. 2b. This is not a surprise, as the main condition that governs these axes is the linear gradient in i (present in both graphs) and has decaying curves near the internal aluminum box.

A slightly more interesting image is a cross section as constant i, shown in fig. 4b. The



(a)



(b)

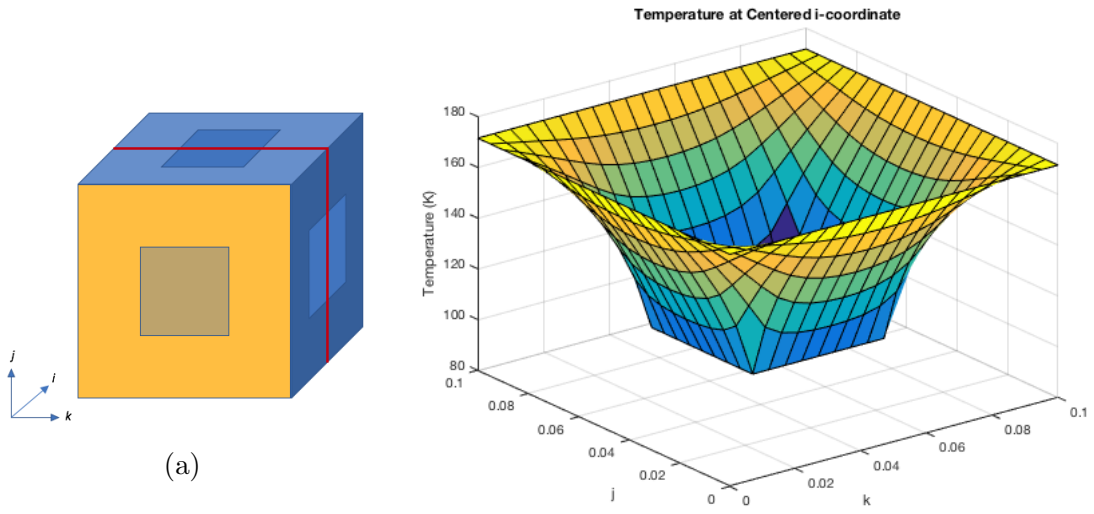
Figure 3: A plot of temperature through a cross-section at the center of the k-axis. There is a noticeable exponential curve towards the aluminum center temperature (100 K), however, the trend through i is linear.

boundaries of the plot all start at the middle of the temperature gradient, then follow our previously shown decay curves to the 100 K aluminum center. This again makes sense as the satellite center acts as a form of heat sink to the heat absorbed by the sun face. Each cross section with constant i takes a similar shape to this, with varying depths of the decay curve as the temperature is closer or further from 100 K.

Overall, the resulting temperature distribution from our system of equations fulfill all expectations from a steady-state system's temperature with two ends held at constant temperatures. The linear gradient between the temperature of the sun-facing side and the temperature of the cosmic microwave background reaches to the inside of the satellite, only altered by the heat-sink-like presence of the aluminum center, which is generally lower in temperature than most of the box interior.

Computational Results

Computationally, both methods produce answers that concur up to 12 decimal places ($10^{-13}\%$ maximum difference). This is due partially to the code written for this assignment using full pivoting. Full pivoting solves matrix equations while ensuring the largest value in the matrix is always moved to the diagonal by shifting rows and columns. This results in less opportunities for numerical error.



(a)

(b)

Figure 4: A plot of temperature through a cross-section at the center of the i-axis. There is a noticeable exponential curve downward from the position in the edge gradient towards the aluminum box center.

Despite these definite equivalencies in numerical solutions, there is a massive disparity in the computation time required for each method. $LAPACK$ is highly optimized for quick computation, and runs at an average of twenty times faster than the basic LU factorization code. For example, with a $11 \times 11 \times 11$ mesh in this system, the LU factorization code takes approximately 19.934 seconds to run, compared to the 5.5×10^{-5} seconds for $LAPACK$, even in double precision accuracy.

Overall, the steady-state behavior of the satellite's temperature behaves as expected: a linear gradient with i away from the sun, and a general temperature decay curve moving inward to the 100 K aluminum center. In the long-term steady-state solution, the system's various materials make no contribution, and the satellite behaves as if it were simply a uniform material with a 100 K center and sides initialized as described above. Provided polysilicon adequately serves its structural and electrical purposes at both 340 K and 2 K, the satellite should be safe in its expected location in space.

The solution to this system using a LU factorization code yields comparable results to those from the highly optimized $LAPACK$ within $10^{-12}\%$. The runtime between two methods illustrates the definite preference for $LAPACK$ due to its nearly twenty-fold speed advantage.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] T.N. Narasimhan. Fourier's heat conduction equation: history, influence, and connections. *Reviews of Geophysics*, February 1999.

[3] W.H. Press. *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, chapter 2.3 "LU Decomposition and Its Applications, pages 34–42. Camridge University Press, 2nd edition, 1992.

[4] Kim Sharp and Franz Matschinsky. Translation of ludwig boltzmanns paper on the relationship between the second fundamental theorem of the mechanical theory of heat and probability calculations regarding the conditions for thermal equilibrium sitzungberichte der kaiserlichen akademie der wissenschaften. mathematisch-naturwissen classe. abt. ii, lxxvi 1877, pp 373-435 (wien. ber. 1877, 76:373-435). reprinted in wiss. abhandlungen, vol. ii, reprint 42, p. 164-223, barth, leipzig, 1909. *Entropy*, 17(4):1971–2009, 2015.

[5] Edward J. Wollack. Wmap big bang cmb test, 5 2016.

Appendix

Attached is the source code for this project, written in Fortran.

```
program steadystateheateq
implicit none
real*8, dimension(:,:), allocatable :: T
real*8, dimension(:), allocatable :: b, x
integer, dimension(:), allocatable :: p, q
integer :: allocatestatus, k, l, m, i, j, meth, iterations, allow,
    alhigh, n = 1, start, finish
real*8 :: Tal = 100, Tsp = 2.725, Tsun = 340.31

n = 10*n+1

allow = int(0.3*n+0.7)
alhigh = int(0.7*n+0.3)

allocate(T(n**3,n**3), STAT = allocatestatus)
if (AllocateStatus /= 0) stop "***Insufficient memory***"
allocate(b(n**3), STAT = allocatestatus)
if (allocatestatus /= 0) stop "***Insufficient memory***"
allocate(x(n**3), STAT = allocatestatus)
if (allocatestatus /= 0) stop "***Insufficient memory***"
allocate(p(n**3), STAT = allocatestatus)
if (allocatestatus /= 0) stop "***Insufficient memory***"
allocate(q(n**3), STAT = allocatestatus)
if (allocatestatus /= 0) stop "***Insufficient memory***"

T = 0
b = 0

! INITIALIZING FACES

do k = 1,n
   do l = 1,n
   T((k-1)*n + l, (k-1)*n + l) = 1
```

```
    b((k-1)*n + l) = Tsun ! i = 1 FACE TO SUN
    T((n-1)*n*n + (k-1)*n + l, (n-1)*n*n + (k-1)*n + l) = 1
    b((n-1)*n*n + (k-1)*n + l) = Tsp ! i = n FACE FARTHEST FROM SUN
    T((k-1)*n*n + l, (k-1)*n*n + l)= 1
    b((k-1)*n*n + l) = (Tsp - Tsun)/(n-1)*(k-1) + Tsun ! GRADIENT
       AWAY FROM SUN, j = 1
    T((k-1)*n*n + (n-1)*n + l, (k-1)*n*n + (n-1)*n + l) = 1
    b((k-1)*n*n + (n-1)*n + l) = (Tsp - Tsun)/(n-1)*(k-1) + Tsun !
       GRADIENT, j = n
    T((k-1)*n*n + (l-1)*n + 1, (k-1)*n*n + (l-1)*n + 1) = 1
    b((k-1)*n*n + (l-1)*n + 1) = (Tsp - Tsun)/(n-1)*(k-1) + Tsun !
       GRADIENT, k = 1
    T((k-1)*n*n + (l-1)*n + n, (k-1)*n*n + (l-1)*n + n) = 1
    b((k-1)*n*n + (l-1)*n + n) = (Tsp - Tsun)/(n-1)*(k-1) + Tsun !
       GRADIENT, k = n
    end do
end do

! FINITE DIFFERENCES WEIGHTING FOR ALL INTERIOR POINTS

do k = 2, n-1
    do l = 2, n-1
       do m = 2, n-1
       T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m) = -6
       T((k-1)*n*n + (l-1)*n + m, (k-1 + 1)*n*n + (l-1)*n + m) = 1
       T((k-1)*n*n + (l-1)*n + m, (k-1 - 1)*n*n + (l-1)*n + m) = 1
       T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1 + 1)*n + m) = 1
       T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1 - 1)*n + m) = 1
       T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m + 1) = 1
       T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m - 1) = 1
       end do
    end do
end do

! ALUMINUM BOX CONDITIONS
```

```
! Removes finite differences placed where unnecessary, reassigns
   as definite solutions

do k = allow,alhigh
   do l = allow,alhigh
      do m = allow,alhigh
      T((k-1)*n*n + (l-1)*n + m, (k-1 + 1)*n*n + (l-1)*n + m) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1 - 1)*n*n + (l-1)*n + m) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1 + 1)*n + m) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1 - 1)*n + m) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m + 1) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m - 1) = 0
      T((k-1)*n*n + (l-1)*n + m, (k-1)*n*n + (l-1)*n + m) = 1
      b((k-1)*n*n + (l-1)*n + m) = Tal
      end do
   end do
end do

print*,'Array initialized...'

!open(unit=100, file = 'array.txt')
!open(unit=200, file = 'b.txt')
!do k = 1,n**3
!   write(100,*) (T(k,l), l = 1,n**3)
!   write(200,*) b(k)
!end do
!close(100)
!close(200)

print*,'Which method?'
print*,'Self-written (1), LAPACK (2)'
read(*,*) meth

if (meth == 1) then
call system_clock(count=start)
call computepaq(size(T,1),T,p,q)
```

```
print*,'PAQ factorization calculated...'
call paqsolve(size(T,1),T,b,p,q,x)
print*,'Solution found!'
call system_clock(count=finish)
print*,'Calculation time:', float(finish-start)/1000
open(unit=300, file = 'ans.txt')
print*,'Writing data...'

do l = 1,n**3
   k = mod(l,n)
   if(k==0) then
      k = n
   end if
   j = mod((l-k)/n + 1, n)
   if(j==0) then
      j = n
   end if
   i = (l-k)/(n**2) + (1-j)/n + 1
   if(i == 0) then
      i = n
   end if
   write(300,*) i, j ,k , x(l)
end do

else if (meth==2) then
call system_clock(count=start)
call dgesv(size(T,1),1,T,size(T,1),p,b,size(b,1),allocatestatus)
call system_clock(count=finish)
print*,'Calculation time:', float(finish-start)/1000
print*,'Solution found!'
open(unit=300, file = 'ans.txt')
print*,'Writing data...'
do l = 1,n**3
   k = mod(l,n)
   if(k==0) then
      k = n
```

```
    end if
    j = mod((l-k)/n + 1, n)
    if(j==0) then
       j = n
    end if
    i = (l-k)/(n**2) + (1-j)/n + 1
    if(i == 0) then
       i = n
    end if
    write(300,*) i, j ,k , b(l)
end do

end if

close(300)
deallocate(T,p,q,b,x)

end program steadystateheateq




subroutine computepaq(n,a,p,q)
double precision :: a(n,n), mult, maxvalue, temp(n)
integer :: n, i, j, k, l, maxrow, maxcol, p(n), q(n)

do i = 1,n
    p(i) = i
    q(i) = i !initialize the row and column trackers
end do
do i = 1,n-1
    maxvalue = abs(a(i,i))
    maxrow = i
    maxcol = i
    do j = i,n
```

```
        do k = i,n
            if (abs(a(j,k)) > maxvalue) then
               maxrow = j
               maxcol = k
               maxvalue = abs(a(j,k))
            end if
        end do
    end do

    if (maxrow > i) then     ! Reorganize a and record shifts in p
        and q
        p(i) = maxrow
        temp(i:n) = a(maxrow,i:n)
        a(maxrow,i:n) = a(i,i:n)
        a(i,i:n) = temp(i:n)
    end if
    if (maxcol > i) then
        q(i) = maxcol
        temp = a(1:n,maxcol)
        a(1:n,maxcol) = a(1:n,i)
        a(1:n,i) = temp(1:n)
    end if

    a(i+1:n,i) = a(i+1:n,i)/a(i,i)    !multipliers in the usual way

    do j = i+1,n
        a(j,i+1:n) = a(j,i+1:n) - a(j,i)*a(i,i+1:n)
    end do
end do

return
end

subroutine paqsolve(n,a,b,p,q,x)
implicit none
double precision :: a(n,n), b(n), w(n), x(n), y(n), s
```

```
integer :: n, i, j, k, p(n), q(n)

! First solve Ly = Pb

y = b

do k = 1,n-1
   if (p(k) > k) then
      s = y(k)
      y(k) = y(p(k))
      y(p(k)) = s
   end if
   do i = k+1,n
      y(i) = y(i) - a(i,k)*y(k)
   end do
end do

! Now solve Uy = Qx

x(n) = y(n)/a(n,n)

do i = n-1,1,-1
   s = y(i)
   do j = i+1,n
      s = s - a(i,j)*x(j)
   end do

   x(i) = s/a(i,i)
end do

! Next properly permute the rows again to solve x = Qy

do k = n-1,1,-1
   if(q(k) > k) then
      s = x(k)
      x(k) = x(q(k))
```

```
        x(q(k)) = s
    end if
end do

return
end
```

MATLAB Plotting Script

```
A = importdata('ans.txt');
dim = size(A);
n = int32(dim(1)^(1/3));
dx = 0.1/(dim(1)^(1/3)-1);



P = zeros(n,n,n);
for i = 1:n
for j = 1:n
for k = 1:n
P(i,j,k) = A((i-1)*n*n + (j-1)*n + k, 4);
end
end
end

[x y] = meshgrid(0:dx:0.1);
mid = ceil(n/2);
surf(x,y,P(:,:,mid));
xlabel('j');
ylabel('i');
zlabel('Temperature (K)');
title('Temperature at Centered k-coordinate');

figure();
surf(x,y,squeeze(P(mid,:,:)));
xlabel('k');
ylabel('j');
```

```
zlabel('Temperature (K)');
title('Temperature at Centered i-coordinate');

figure();
surf(x,y,squeeze(P(:,mid,:)));
xlabel('k');
ylabel('i');
zlabel('Temperature (K)');
title('Temperature at Centered j-coordinate');


filename='Tempdistr'
limits = [0;0.1;0;0.1;2;350];
h = figure;
for i = 1:n
%figure()
surf(x,y,squeeze(P(i,:,:)));
title(num2str(i))
axis(limits)
drawnow
% Capture the plot as an image
frame = getframe(h);
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);
% Write to the GIF File
if i == 1
imwrite(imind,cm,filename,'gif', 'Loopcount',inf,'DelayTime',0.05)
    ;
else
imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime'
    ,0.05);
end
end
```