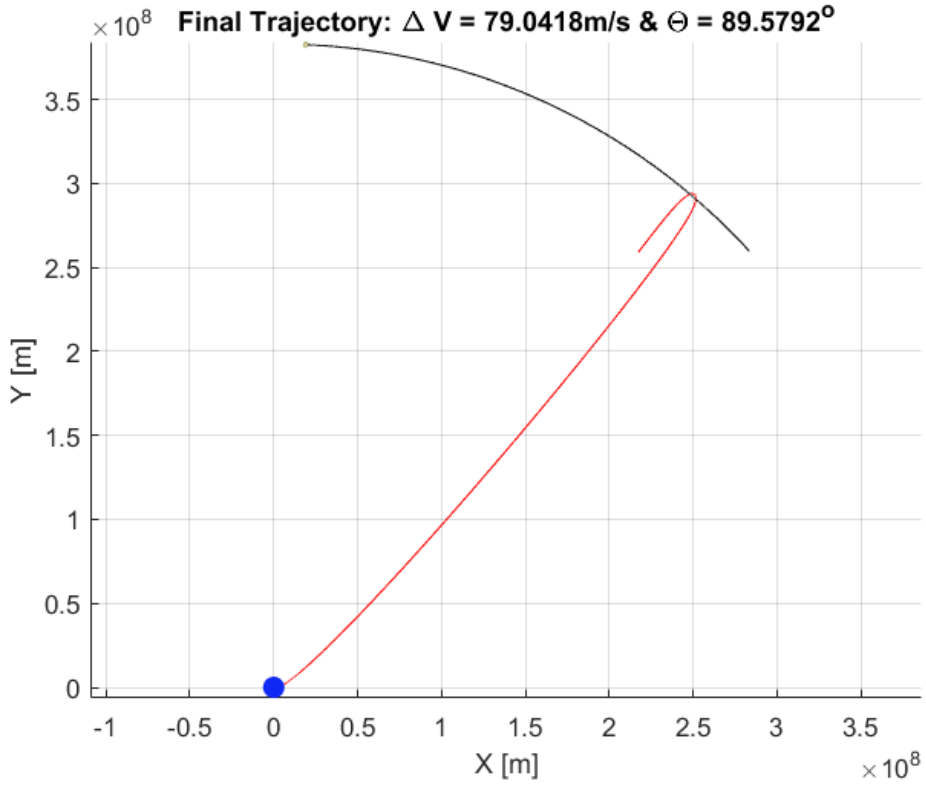
Assignment 2 Report
Aaron Aboaf, Brandon Sundahl
ASEN 4057 -- 2/9/2018

The most important results from the matlab profiler are the self times and the calls. The self times are important because this is how much time is used per each function. The number of calls is important as well because the functions can run faster and easier if the number of calls is minimized. If the functions listed on the profiler are called more, then the self time increases, leading to slower performance.

Less calls to the fmincon function will improve runtime. The code runs fmincon 7 times in order to optimize the results. If a seperate function held all these calls, the overall runtime would improve. As shown in our code, multiple functions performing calculations more than once will improve performance as well.

Another strategy that can improve runtime could be parallelization. During the code development, a script was written to iterate through a range of deltaVs and thetas. This “brute force” method involves hundreds of calls to ode45 alone making for slow progress. Since the inputs to the ode45 function can be set prior to the integration, it would be possible to parallelize the code using a parfor loop. This would effectively reduce the time required to compute the individual trajectories.

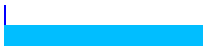
As you can see below through the profiler, the code that was ran produced the plot below. The results are as follows. In order to minimize the deltaV to reach the Earth, the spacecraft must fire a chance in velocity of 79.0418 m/s at 89.5792 degrees. Note this is almost all in the Y direction. The trajectory of the spacecraft with this burn is shown below. Results of the profiling report follow.



Some notes from the profiler:
The profiler returns information about the function calls from the main script. According to the profiler, the two functions that the program spent the most time in were ode45() and gforce(). This is expected as these are the backbones of the calculations that are required to simulate the spacecraft trajectory. The following pages show the overall summary for the minimization script and then the summaries for the ode45() function and gforce() function on their own.

Profile Summary

Generated 09-Feb-2018 23:04:36 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
main_fmincon	1	17.195 s	0.191 s	
ode45	1088	15.321 s	4.324 s	
fmincon	7	9.078 s	0.037 s	
barrier	7	8.849 s	0.396 s	
...tFunction(vec(1),vec(2),smooth,P_end)	1027	7.818 s	0.007 s	
OptFunction	1027	7.811 s	0.074 s	
orbit_run	60	7.620 s	0.006 s	
orbit_equations	379568	6.810 s	2.717 s	
gforce	1138704	4.093 s	2.865 s	
...n>@(t,traj)orbit_equations(t,traj)	186337	3.807 s	0.442 s	
...n>@(t,traj)orbit_equations(t,traj)	188976	3.800 s	0.438 s	
computeFinDiffGradAndJac	100	2.766 s	0.010 s	
finitedifferences	100	2.756 s	0.023 s	
finDiffEvalAndChkErr	200	2.730 s	0.007 s	

funfun\private\odezero	53018	1.929 s	1.139 s	
euclidian_distance	1388908	1.500 s	1.500 s	
funfun\private\ntrp45	61464	1.093 s	1.093 s	
myevents	62551	0.733 s	0.461 s	
optim\private\computeTrialStep	820	0.390 s	0.100 s	
funfun\private\odearguments	1088	0.202 s	0.080 s	
odeset	1088	0.197 s	0.160 s	
optim\private\tangentialStep	454	0.165 s	0.056 s	
odeget	11968	0.164 s	0.067 s	
...n>@(t,traj)orbit_equations(t,traj)	4255	0.097 s	0.013 s	
odeget>getknownfield	11968	0.097 s	0.097 s	
optim\private\projConjGrad	454	0.090 s	0.036 s	
optim\private\normalStep	454	0.076 s	0.013 s	
funfun\private\odefinalize	1088	0.076 s	0.076 s	
optim\private\solveAugSystem	1064	0.073 s	0.025 s	
optim\private\backsolveSys	1161	0.053 s	0.053 s	
earth_sphere	1	0.052 s	0.011 s	

optim\private\updatePenaltyParam	820	0.051 s	0.041 s	
optim\private\normalNewtonStep	454	0.050 s	0.014 s	
optimget	273	0.049 s	0.006 s	
optim\private\nlpStopTest	100	0.048 s	0.042 s	
funfun\private\odeevents	1088	0.047 s	0.018 s	
...e\projConjGrad>computeProjResidual	454	0.045 s	0.011 s	
getIpOptions	7	0.044 s	0.011 s	
optimget>optimgetfast	273	0.043 s	0.043 s	
strmatch	2176	0.037 s	0.037 s	
barrier>nlpInterfaceFcn	834	0.032 s	0.032 s	
createExitMsg	7	0.030 s	0.030 s	
optim\private\formAndFactorKKTmatrix	97	0.026 s	0.003 s	
funfun\private\odemass	1088	0.026 s	0.012 s	
optim\private\formConstraints	827	0.024 s	0.024 s	
optim\private\xFixedAndBounds	820	0.023 s	0.023 s	
axis	1	0.022 s	0.005 s	
optim\private\formJacobian	100	0.022 s	0.015 s	

optim\private\leastSquaresLagrangeMults	156	0.020 s	0.006 s	
optim\private\hessTimesVector	1728	0.019 s	0.019 s	
optim\private\formAndFactorAugMatrix	91	0.018 s	0.003 s	
spdiags	482	0.017 s	0.017 s	
moon_sphere	1	0.017 s	0.006 s	
...rmAndFactorKKTmatrix>formKKTmatrix	97	0.016 s	0.008 s	
optim\private\acceptanceTest	820	0.016 s	0.016 s	
optim\private\truncateTangStep	454	0.014 s	0.008 s	
optim\private\normalCauchyStep	454	0.014 s	0.014 s	
title	1	0.011 s	0.009 s	
axis>LocSetEqual	1	0.009 s	0.008 s	
newplot	3	0.009 s	0.004 s	
view	2	0.009 s	0.004 s	
hold	1	0.008 s	0.005 s	
getNumericOrStringFieldValue	28	0.008 s	0.002 s	
...te\projConjGrad>getProjectionAngle	454	0.008 s	0.008 s	
axis>isnumericAxes	1	0.008 s	0.008 s	

...rmAndFactorAugMatrix>formAugMatrix	91	0.008 s	0.005 s	
close	1	0.008 s	0.003 s	
optim\private\solveKKTsystem	97	0.007 s	0.002 s	
optim\private\computeHessian	100	0.007 s	0.006 s	
...AndFactorKKTmatrix>factorKKTmatrix	97	0.007 s	0.007 s	
...AndFactorAugMatrix>factorAugMatrix	91	0.007 s	0.007 s	
formLambdaStruct	7	0.007 s	0.007 s	
optim\private\barrierTestAndUpdate	97	0.007 s	0.002 s	
opt...ivate\fractionToBoundaryTangential	454	0.006 s	0.006 s	
num2str	2	0.006 s	0.003 s	
optimfcnchk	7	0.006 s	0.003 s	
barrier>initialization	7	0.005 s	0.005 s	
colormap	2	0.005 s	0.005 s	
...\projConjGrad>stplngthToTrBoundary	454	0.005 s	0.005 s	
...rrierTestAndUpdate>barrierStopTest	97	0.005 s	0.005 s	
optim\private\fractionToBoundaryScaled	551	0.005 s	0.005 s	
prepareOptionsForSolver	7	0.005 s	0.001 s	

zlabel	1	0.005 s	0.004 s	
close>safegetchildren	1	0.004 s	0.000 s	
setSizes	7	0.004 s	0.004 s	
optimlib\private\classifyBoundsOnVars	7	0.004 s	0.003 s	
createOptionFeedback	7	0.004 s	0.004 s	
shading	1	0.004 s	0.004 s	
opt...vate\fractionToBoundaryHonorBounds	97	0.004 s	0.004 s	
allchild	1	0.004 s	0.002 s	
xlabel	1	0.004 s	0.004 s	
ylabel	1	0.004 s	0.004 s	
view>ViewCore	2	0.004 s	0.003 s	
isoptimargdbl	14	0.003 s	0.003 s	
perturbation	1079	0.003 s	0.003 s	
setOptimFcnHandleOnWorkers	7	0.003 s	0.002 s	
validateFinDiffRelStep	7	0.003 s	0.001 s	
optim\private\backtrack	269	0.003 s	0.003 s	
markFigure	7	0.003 s	0.003 s	

num2str>handleNumericPrecision	2	0.003 s	0.001 s	
axescheck	3	0.003 s	0.003 s	
optim\private\fractionToBoundary	97	0.002 s	0.002 s	
gobjects	6	0.002 s	0.002 s	
fcnchk	7	0.002 s	0.002 s	
num2str>convertUsingRecycledSprintf	2	0.002 s	0.002 s	
validateopts_UseParallel	107	0.002 s	0.002 s	
nonzerosign	200	0.002 s	0.002 s	
checkbounds	7	0.002 s	0.002 s	
graph2d\private\labelcheck	4	0.002 s	0.002 s	
newplot>ObserveAxesNextPlot	3	0.002 s	0.002 s	
optim\private\accStepTRupdate	93	0.002 s	0.002 s	
fwdFinDiffInsideBnds	200	0.002 s	0.002 s	
grid	1	0.001 s	0.001 s	
linspace	2	0.001 s	0.001 s	
optim\private\dampingProcedure	93	0.001 s	0.001 s	
getpixelposition	1	0.001 s	0.000 s	

view>isAxesHandle	2	0.001 s	0.001 s	
newplot>ObserveFigureNextPlot	3	0.001 s	0.001 s	
uitools\private\getPixelPositionHelper	1	0.001 s	0.001 s	
checkoptionsize	7	0.001 s	0.001 s	
...te\classifyBoundsOnVars>equalFloat	7	0.001 s	0.001 s	
barrier>printLevel	7	0.001 s	0.001 s	
plotedit	1	0.001 s	0.001 s	
gray	1	0.001 s	0.001 s	
onCleanup>onCleanup.delete	1	0.001 s	0.000 s	
ishold	1	0.000 s	0.000 s	
...set(rootobj,'ShowHiddenHandles',Temp)	1	0.000 s	0.000 s	
uitools\private\allchildRootHelper	1	0.000 s	0.000 s	
allchild>getchildren	1	0.000 s	0.000 s	
axis>iscartesian	1	0.000 s	0.000 s	
onCleanup>onCleanup.onCleanup	1	0.000 s	0.000 s	
close>getEmptyHandleList	1	0.000 s	0.000 s	
barrier>displayHeader	7	0.000 s	0.000 s	

axis>allAxes	1	0.000 s	0.000 s	
close>request_close	1	0.000 s	0.000 s	
close>checkfigs	1	0.000 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

ode45 (Calls: 1088, Time: 15.321 s)

Generated 09-Feb-2018 23:06:25 using performance time.
function in file D:\Matlab2016b\toolbox\matlab\funfun\ode45.m
Copy to new window for comparing multiple runs

Refresh

☑ Show parent functions ☑ Show busy lines ☑ Show child functions

☑ Show Code Analyzer results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
orbit_run	function	60
OptFunction	function	1027
main_fmincon	script	1

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
353	`odezero(@ntrp45,eventFcn,event...`	53018	2.389 s	15.6%	
261	`f(:,2) = feval(odeFcn,t+hA(1),...`	63080	1.945 s	12.7%	
262	`f(:,3) = feval(odeFcn,t+hA(2),...`	63080	1.677 s	10.9%	
263	`f(:,4) = feval(odeFcn,t+hA(3),...`	63080	1.631 s	10.6%	
265	`f(:,6) = feval(odeFcn,t+hA(5),...`	63080	1.624 s	10.6%	
All other lines			6.057 s	39.5%	
Totals			15.321 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot

...n>@(t,traj)orbit_equations(t,traj)	anonymous function	188916	3.794 s	24.8%	
...n>@(t,traj)orbit_equations(t,traj)	anonymous function	185310	3.778 s	24.7%	
funfun\private\odezero	function	53018	1.929 s	12.6%	
funfun\private\ntrp45	function	54106	1.000 s	6.5%	
funfun\private\odearguments	function	1088	0.202 s	1.3%	
...n>@(t,traj)orbit_equations(t,traj)	anonymous function	4254	0.096 s	0.6%	
funfun\private\odefinalize	function	1088	0.076 s	0.5%	
odeget	function	4352	0.049 s	0.3%	
funfun\private\odeevents	function	1088	0.047 s	0.3%	
funfun\private\odemass	function	1088	0.026 s	0.2%	
Self time (built-ins, overhead, etc.)			4.324 s	28.2%	
Totals			15.321 s	100%	

Code Analyzer results

Line number	Message
356	The variable 'teout' appears to change size on every loop iteration. Consider preallocating for speed.
357	The variable 'yeout' appears to change size on every loop iteration. Consider preallocating for speed.
358	The variable 'ieout' appears to change size on every loop iteration. Consider preallocating for speed.
378	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
379	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
382	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.

,varargin)
 medium order method.
SPAN = [T0 TFINAL] integrates
(t,y) from time T0 to TFINAL
ction handle. For a scalar T
olumn vector corresponding
OUT corresponds to a time
ain solutions at specific
ll decreasing), use TSPAN =

) solves as above with default
n OPTIONS, an argument created
tails. Commonly used options
' (1e-3 by default) and vector
components 1e-6 by default).
be non-negative, use
 the indices of these

 with mass matrix M that is
roperty to a function handle
mass matrix. If the mass matrix
value of the 'Mass' option. If
e variable Y and the function
t T, set 'MStateDependence' to
s with singular mass matrices.

),OPTIONS) with the 'Events'
e EVENTS, solves as above
, called event functions,
ther the integration is
ction of the zero crossing
 returned by EVENTS:
 For the I-th event function:
RMINAL(I)=1 if the integration
ction and 0 otherwise.
ted (the default), +1 if only
g, and -1 if only zeros where
 is a column vector of times
orresponding solutions, and

383	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
384	The variable 'f3d' appears to change size on every loop iteration. Consider preallocating for speed.
406	The variable 'tout_new' appears to change size on every loop iteration. Consider preallocating for speed.
407	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
412	The variable 'tout_new' appears to change size on every loop iteration. Consider preallocating for speed.
414	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
416	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
427	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
428	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
431	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
432	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.

Coverage results

Show coverage for parent directory

Total lines in function	482
Non-code lines (comments, blank lines)	154
Code lines (lines that can run)	328
Code lines that did run	185
Code lines that did not run	143
Coverage (did run/can run)	56.40 %

Function listing

Color highlight code according to time

time **Calls** **line**

curred.

is a structure that can be
its first derivative at
chosen by ODE45 are returned
umn SOL.y(:,I) contains
ected, SOL.xe is a row vector
of SOL.ye are the corresponding
cify which event occurred.

he default relative error
olerance of 1e-6 for each
of the solution.

result of ODEFUN(T,Y):

23T, ODE23TB, ODE15I,
3, ODEPRINT, DEVAL,
BITODE, FUNCTION_HANDLE.

Runge-Kutta (4,5) pair of
FM, DOPRI5, DP(4,5) and DP54.
municated privately by
done.

uite, L. F. Shampine and
Computing, 18-1, 1997.

6-14-94

```
function varargout = ode45(ode,tspan,y0,options,
%ODE45  Solve non-stiff differential equations,
%   [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TS
%   the system of differential equations y' = f
%   with initial conditions Y0. ODEFUN is a func
%   and a vector Y, ODEFUN(T,Y) must return a co
%   to f(t,y). Each row in the solution array YO
%   returned in the column vector TOUT.  To obta
%   times T0,T1,...,TFINAL (all increasing or al
%   [T0 T1 ... TFINAL].
%
%   [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS)
%   integration properties replaced by values in
%   with the ODESET function. See ODESET for det
%   are scalar relative error tolerance 'RelTol
%   of absolute error tolerances 'AbsTol' (all c
%   If certain components of the solution must b
%   ODESET to set the 'NonNegative' property to
%   components.
%
%   ODE45 can solve problems M(t,y)*y' = f(t,y)
%   nonsingular. Use ODESET to set the 'Mass' pr
%   MASS if MASS(T,Y) returns the value of the m
%   is constant, the matrix can be used as the v
%   the mass matrix does not depend on the state
%   MASS is to be called with one input argument
%   'none'. ODE15S and ODE23T can solve problems
%
%   [TOUT,YOUT,TE,YE,IE] = ODE45(ODEFUN,TSPAN,YO
%   property in OPTIONS set to a function handle
%   while also finding where functions of (T,Y),
%   are zero. For each function you specify whet
%   to terminate at a zero and whether the direc
%   matters. These are the three column vectors
%   [VALUE,ISTERMINAL,DIRECTION] = EVENTS(T,Y).
%   VALUE(I) is the value of the function, ISTER
%   is to terminate at a zero of this event func
%   DIRECTION(I)=0 if all zeros are to be comput
%   zeros where the event function is increasing
%   the event function is decreasing. Output TE
%   at which events occur. Rows of YE are the co
```

puts'));

```
    % sol = odeXX(...)
[t,y,...] = odeXX(...)
```

```
, f0, odeArgs, odeFcn, ...
nmax, htry, htspan, dataType] = ...
de, tspan, y0, options, varargin);
```

```
st');
```

```
pt,'fast');
```

```
                    42  %   indices in vector IE specify which event occ
                    43  %
                    44  %   SOL = ODE45(ODEFUN,[T0 TFINAL],Y0...) retur
                    45  %   used with DEVAL to evaluate the solution or
                    46  %   any point between T0 and TFINAL. The steps c
                    47  %   in a row vector SOL.x.  For each I, the colu
                    48  %   the solution at SOL.x(I). If events were det
                    49  %   of points at which events occurred. Columns
                    50  %   solutions, and indices in vector SOL.ie spec
                    51  %
                    52  %   Example
                    53  %         [t,y]=ode45(@vdp1,[0 20],[2 0]);
                    54  %         plot(t,y(:,1));
                    55  %     solves the system y' = vdp1(t,y), using th
                    56  %     tolerance 1e-3 and the default absolute to
                    57  %     component, and plots the first component c
                    58  %
                    59  %   Class support for inputs TSPAN, Y0, and the
                    60  %     float: double, single
                    61  %
                    62  %   See also ODE23, ODE113, ODE15S, ODE23S, ODE2
                    63  %            ODESET, ODEPLOT, ODEPHAS2, ODEPHAS3
                    64  %            ODEEXAMPLES, RIGIDODE, BALLODE, ORB
                    65
                    66  %   ODE45 is an implementation of the explicit R
                    67  %   Dormand and Prince called variously RK5(4)7B
                    68  %   It uses a "free" interpolant of order 4 comm
                    69  %   Dormand and Prince.  Local extrapolation is
                    70
                    71  %   Details are to be found in The MATLAB ODE Su
                    72  %   M. W. Reichelt, SIAM Journal on Scientific C
                    73
                    74  %   Mark W. Reichelt and Lawrence F. Shampine, 6
                    75  %   Copyright 1984-2011 The MathWorks, Inc.
                    76
< 0.01     1088     77  solver_name = 'ode45';
                    78
                    79  % Check inputs
< 0.01     1088     80  if nargin < 4
                    81    options = [];
                    82    if nargin < 3
```

```
ast');

t arguments to outputFcn.


));

t only at tspan points

ted points, no refinement

ted points, with refinement


','fast'),'on');


ut,ieout] = ...
ns,varargin);


'cn,t0,y0,options,varargin);

','fast');

ssumedNo'));

'));

 odeArgs.
Jsed,Mtype,odeFcn,odeArgs,Mfun,M);
```

```
                    83      y0 = [];
                    84      if nargin < 2
                    85        tspan = [];
                    86        if nargin < 1
                    87          error(message('MATLAB:ode45:NotEnoughInp
                    88        end
                    89      end
                    90    end
                    91  end
                    92
                    93  % Stats
< 0.01      1088    94  nsteps  = 0;
< 0.01      1088    95  nfailed = 0;
< 0.01      1088    96  nfevals = 0;
                    97
                    98  % Output
< 0.01      1088    99  FcnHandlesUsed  = isa(ode,'function_handle');
< 0.01      1088   100  output_sol = (FcnHandlesUsed && (nargout==1));
< 0.01      1088   101  output_ty  = (~output_sol && (nargout > 0));  %
                   102  % There might be no output requested...
                   103
< 0.01      1088   104  sol = []; f3d = [];
< 0.01      1088   105  if output_sol
                   106    sol.solver = solver_name;
                   107    sol.extdata.odefun = ode;
                   108    sol.extdata.options = options;
                   109    sol.extdata.varargin = varargin;
                   110  end
                   111
                   112  % Handle solver arguments
  0.21      1088   113  [neq, tspan, ntspan, next, t0, tfinal, tdir, y0,
            1088   114   options, threshold, rtol, normcontrol, normy, h
            1088   115      odearguments(FcnHandlesUsed, solver_name, o
< 0.01      1088   116  nfevals = nfevals + 1;
                   117
                   118  % Handle the output
< 0.01      1088   119  if nargout > 0
  0.02      1088   120    outputFcn = odeget(options,'OutputFcn',[],'fa
                   121  else
                   122    outputFcn = odeget(options,'OutputFcn',@odeplo
< 0.01      1088   123  end
```

],'fast');

n
e(odeFcn,y0,threshold,idxNonNegative);

r((2^11)/neq));

put only at tspan points

oc in chunks
oor((2^13)/neq));

```
< 0.01     1088  124  outputArgs = {};
< 0.01     1088  125  if isempty(outputFcn)
< 0.01     1088  126    haveOutputFcn = false;
                 127  else
                 128    haveOutputFcn = true;
                 129    outputs = odeget(options,'OutputSel',1:neq,'fa
                 130    if isa(outputFcn,'function_handle')
                 131      % With MATLAB 6 syntax pass additional input
                 132      outputArgs = varargin;
                 133    end
< 0.01     1088  134  end
  0.01     1088  135  refine = max(1, odeget(options,'Refine',4,'fast'
< 0.01     1088  136  if ntspan > 2
                 137    outputAt = 'RequestedPoints';          % output
< 0.01     1088  138  elseif refine <= 1
                 139    outputAt = 'SolverSteps';              % comput
< 0.01     1088  140  else
< 0.01     1088  141    outputAt = 'RefinedSteps';             % comput
< 0.01     1088  142    S = (1:refine-1) / refine;
< 0.01     1088  143  end
  0.02     1088  144  printstats = strcmp(odeget(options,'Stats','off
                 145
                 146  % Handle the event function
  0.05     1088  147  [haveEventFcn,eventFcn,eventArgs,valt,teout,yeou
           1088  148      odeevents(FcnHandlesUsed,odeFcn,t0,y0,optio
                 149
                 150  % Handle the mass matrix
  0.03     1088  151  [Mtype, M, Mfun] =  odemass(FcnHandlesUsed,odeF
< 0.01     1088  152  if Mtype > 0  % non-trivial mass matrix
                 153    Msingular = odeget(options,'MassSingular','no
                 154    if strcmp(Msingular,'maybe')
                 155      warning(message('MATLAB:ode45:MassSingularAs
                 156    elseif strcmp(Msingular,'yes')
                 157      error(message('MATLAB:ode45:MassSingularYes
                 158    end
                 159    % Incorporate the mass matrix into odeFcn and
                 160    [odeFcn,odeArgs] = odemassexplicit(FcnHandlesU
                 161    f0 = feval(odeFcn,t0,y0,odeArgs{:});
                 162    nfevals = nfevals + 1;
                 163  end
                 164
```

```
 -355/33          0
 46732/5247       500/1113
 49/176           125/192
 -5103/18656      -2187/6784
 0                11/84
 0                0

9200; 22/525; -1/40];
```

```
.8 * rtol^pow);
```

```
/ (0.8 * rtol^pow);
```

```
outputArgs{:});
```

```
t t+hmin is only slightly
```

```
n't limit absh until new hmin
```

```
                 165  % Non-negative solution components
  0.01     1088  166  idxNonNegative = odeget(options,'NonNegative',[
< 0.01     1088  167  nonNegative = ~isempty(idxNonNegative);
< 0.01     1088  168  if nonNegative  % modify the derivative functio
                 169    [odeFcn,thresholdNonNegative] = odenonnegative
                 170    f0 = feval(odeFcn,t0,y0,odeArgs{:});
                 171    nfevals = nfevals + 1;
                 172  end
                 173
< 0.01     1088  174  t = t0;
< 0.01     1088  175  y = y0;
                 176
                 177  % Allocate memory if we're generating output.
< 0.01     1088  178  nout = 0;
< 0.01     1088  179  tout = []; yout = [];
< 0.01     1088  180  if nargout > 0
< 0.01     1088  181    if output_sol
                 182      chunk = min(max(100,50*refine), refine+floo
                 183      tout = zeros(1,chunk,dataType);
                 184      yout = zeros(neq,chunk,dataType);
                 185      f3d  = zeros(neq,7,chunk,dataType);
< 0.01     1088  186    else
< 0.01     1088  187      if ntspan > 2                         % outp
                 188        tout = zeros(1,ntspan,dataType);
                 189        yout = zeros(neq,ntspan,dataType);
< 0.01     1088  190      else                                  % allo
< 0.01     1088  191        chunk = min(max(100,50*refine), refine+flo
< 0.01     1088  192        tout = zeros(1,chunk,dataType);
< 0.01     1088  193        yout = zeros(neq,chunk,dataType);
< 0.01     1088  194      end
< 0.01     1088  195    end
< 0.01     1088  196    nout = 1;
< 0.01     1088  197    tout(nout) = t;
< 0.01     1088  198    yout(:,nout) = y;
< 0.01     1088  199  end
                 200
                 201  % Initialize method parameters.
< 0.01     1088  202  pow = 1/5;
< 0.01     1088  203  A = [1/5, 3/10, 4/5, 8/9, 1, 1];
< 0.01     1088  204  B = [
                 205      1/5          3/40     44/45    19372/6561
```

iled attempts

```
deArgs{:});
deArgs{:});
deArgs{:});
deArgs{:});
deArgs{:});
```

```
;
```

```
);

ew(idxNonNegative)<0)
e)) ) / errwt ;
```

```
                206      0           9/40    -56/15  -25360/2187
                207      0           0       32/9    64448/6561
                208      0           0       0       -212/729
                209      0           0       0       0
                210      0           0       0       0
                211      0           0       0       0
                212      ];
< 0.01     1088  213  E = [71/57600; 0; -71/16695; 71/1920; -17253/33
< 0.01     1088  214  f = zeros(neq,7,dataType);
< 0.01     1088  215  hmin = 16*eps(t);
< 0.01     1088  216  if isempty(htry)
                217    % Compute an initial step size h using y'(t).
< 0.01     1088  218    absh = min(hmax, htspan);
< 0.01     1088  219    if normcontrol
                220      rh = (norm(f0) / max(normy,threshold)) / (0
< 0.01     1088  221    else
< 0.01     1088  222      rh = norm(f0 ./ max(abs(y),threshold),inf) /
< 0.01     1088  223    end
< 0.01     1088  224    if absh * rh > 1
< 0.01     1088  225      absh = 1 / rh;
< 0.01     1088  226    end
< 0.01     1088  227    absh = max(absh, hmin);
                228  else
                229    absh = min(hmax, max(hmin, htry));
< 0.01     1088  230  end
< 0.01     1088  231  f(:,1) = f0;
                232
                233  % Initialize the output function.
< 0.01     1088  234  if haveOutputFcn
                235    feval(outputFcn,[t tfinal],y(outputs),'init',
                236  end
                237
                238  % THE MAIN LOOP
                239
< 0.01     1088  240  done = false;
< 0.01     1088  241  while ~done
                242
                243    % By default, hmin is a small number such that
                244    % different than t.  It might be 0 if t is 0.
< 0.01    53018  245    hmin = 16*eps(t);
< 0.01    53018  246    absh = min(hmax, max(hmin, absh));    % could
```

```
y),abs(ynew)),threshold),inf);
ew(idxNonNegative)<0)
e)) ./ thresholdNonNegative, inf);
```

```
error is no more than the
yield an error of rtol on
his step, as the case may be,
es.
d step
```

```
onTolNotMet', sprintf( '%e', t ), sprintf( '%e', hmin )));
, sol,...
outputArgs,...
 [nsteps, nfailed, nfevals],...
 yout,...
n, teout, yeout, ieout,...
Negative});
```

```
(rtol/err)^pow));
```

```
< 0.01    53018   247   h = tdir * absh;
                  248
                  249   % Stretch the step if within 10% of tfinal-t.
< 0.01    53018   250   if 1.1*absh >= abs(tfinal - t)
                  251     h = tfinal - t;
                  252     absh = abs(h);
                  253     done = true;
                  254   end
                  255
                  256   % LOOP FOR ADVANCING ONE STEP.
< 0.01    53018   257   nofailed = true;                          % no fa
< 0.01    53018   258   while true
  0.01    63080   259     hA = h * A;
  0.01    63080   260     hB = h * B;
  1.94    63080   261     f(:,2) = feval(odeFcn,t+hA(1),y+f*hB(:,1),od
  1.68    63080   262     f(:,3) = feval(odeFcn,t+hA(2),y+f*hB(:,2),od
  1.63    63080   263     f(:,4) = feval(odeFcn,t+hA(3),y+f*hB(:,3),od
  1.62    63080   264     f(:,5) = feval(odeFcn,t+hA(4),y+f*hB(:,4),od
  1.62    63080   265     f(:,6) = feval(odeFcn,t+hA(5),y+f*hB(:,5),od
                  266
< 0.01    63080   267     tnew = t + hA(6);
< 0.01    63080   268     if done
                  269       tnew = tfinal;   % Hit end point exactly.
                  270     end
< 0.01    63080   271     h = tnew - t;      % Purify h.
                  272
  0.15    63080   273     ynew = y + f*hB(:,6);
  1.49    63080   274     f(:,7) = feval(odeFcn,tnew,ynew,odeArgs{:});
< 0.01    63080   275     nfevals = nfevals + 6;
                  276
                  277     % Estimate the error.
< 0.01    63080   278     NNrejectStep = false;
< 0.01    63080   279     if normcontrol
                  280       normynew = norm(ynew);
                  281       errwt = max(max(normy,normynew),threshold)
                  282       err = absh * (norm(f * E) / errwt);
                  283       if nonNegative && (err <= rtol) && any(yne
                  284         errNN = norm( max(0,-ynew(idxNonNegative
                  285         if errNN > rtol
                  286           err = errNN;
                  287           NNrejectStep = true;
```

```
ssful step

) <0)
egative),0);

,t,y,tnew,ynew,t0,h,f,idxNonNegative);

l event.
e(end)].

rpolating polynomial.

f,idxNonNegative);
```

```
                    288         end
                    289       end
< 0.01    63080     290     else
  0.23    63080     291       err = absh * norm((f * E) ./ max(max(abs(y
< 0.01    63080     292       if nonNegative && (err <= rtol) && any(yne
                    293         errNN = norm( max(0,-ynew(idxNonNegative
                    294         if errNN > rtol
                    295           err = errNN;
                    296           NNrejectStep = true;
                    297         end
                    298       end
< 0.01    63080     299     end
                    300
                    301     % Accept the solution only if the weighted e
                    302     % tolerance rtol.  Estimate an h that will y
                    303     % the next step or the next try at taking th
                    304     % and use 0.8 of this value to avoid failure
< 0.01    63080     305     if err > rtol                         % Failed
< 0.01    10062     306       nfailed = nfailed + 1;
< 0.01    10062     307       if absh <= hmin
                    308         warning(message('MATLAB:ode45:Integratio
                    309         solver_output = odefinalize(solver_name,
                    310                                     outputFcn, o
                    311                                     printstats,
                    312                                     nout, tout,
                    313                                     haveEventFcn
                    314                                     {f3d,idxNonN
                    315         if nargout > 0
                    316           varargout = solver_output;
                    317         end
                    318         return;
                    319       end
                    320
< 0.01    10062     321       if nofailed
< 0.01    10062     322         nofailed = false;
< 0.01    10062     323         if NNrejectStep
                    324           absh = max(hmin, 0.5*absh);
< 0.01    10062     325         else
< 0.01    10062     326           absh = max(hmin, absh * max(0.1, 0.8*
< 0.01    10062     327         end
                    328       else
```

```
% requires chunk >= refine

pe));
```

```
s, no refinement
```

```
s, with refinement
```

```
NonNegative), ynew];
 tspan points
```

```
stop,ystop
```

```
                329          absh = max(hmin, 0.5 * absh);
< 0.01    10062  330        end
< 0.01    10062  331        h = tdir * absh;
< 0.01    10062  332        done = false;
                333
< 0.01    53018  334      else                                % Succes
                335
< 0.01    53018  336        NNreset_f7 = false;
< 0.01    53018  337        if nonNegative && any(ynew(idxNonNegative)
                338          ynew(idxNonNegative) = max(ynew(idxNonNe
                339          if normcontrol
                340            normynew = norm(ynew);
                341          end
                342          NNreset_f7 = true;
                343        end
                344
< 0.01    53018  345        break;
                346
< 0.01    10062  347      end
< 0.01    10062  348    end
< 0.01    53018  349    nsteps = nsteps + 1;
                350
< 0.01    53018  351    if haveEventFcn
  2.39    53018  352      [te,ye,ie,valt,stop] = ...
          53018  353          odezero(@ntrp45,eventFcn,eventArgs,valt
< 0.01    53018  354      if ~isempty(te)
< 0.01     1088  355        if output_sol || (nargout > 2)
< 0.01     1088  356          teout = [teout, te];
< 0.01     1088  357          yeout = [yeout, ye];
< 0.01     1088  358          ieout = [ieout, ie];
< 0.01     1088  359        end
< 0.01     1088  360        if stop               % Stop on a termina
                361          % Adjust the interpolation data to [t te
                362
                363          % Update the derivatives using the inte
< 0.01     1088  364          taux = t + (te(end) - t)*A;
  0.04     1088  365          [~,f(:,2:7)] = ntrp45(taux,t,y,[],[],h,
                366
< 0.01     1088  367          tnew = te(end);
< 0.01     1088  368          ynew = ye(:,end);
< 0.01     1088  369          h = tnew - t;
```

```
xt),t,y,[],[],h,f,idxNonNegative)];
```

```
];  % requires chunk >= refine
e)];
```

```
w(outputs,:),'',outputArgs{:});
```

```
: err may be 0.
```

```
< 0.01     1088   370          done = true;
< 0.01     1088   371        end
< 0.01     1088   372      end
< 0.01    53018   373    end
                  374
< 0.01    53018   375    if output_sol
                  376      nout = nout + 1;
                  377      if nout > length(tout)
                  378        tout = [tout, zeros(1,chunk,dataType)];
                  379        yout = [yout, zeros(neq,chunk,dataType)];
                  380        f3d  = cat(3,f3d,zeros(neq,7,chunk,dataTyp
                  381      end
                  382      tout(nout) = tnew;
                  383      yout(:,nout) = ynew;
                  384      f3d(:,:,nout) = f;
                  385    end
                  386
  0.01    53018   387    if output_ty || haveOutputFcn
< 0.01    53018   388      switch outputAt
  0.10    53018   389       case 'SolverSteps'        % computed points
                  390        nout_new = 1;
                  391        tout_new = tnew;
                  392        yout_new = ynew;
  0.04    53018   393       case 'RefinedSteps'       % computed points
  0.01    53018   394        tref = t + (tnew-t)*S;
< 0.01    53018   395        nout_new = refine;
  0.17    53018   396        tout_new = [tref, tnew];
  1.21    53018   397        yout_new = [ntrp45(tref,t,y,[],[],h,f,idx
                  398       case 'RequestedPoints'    % output only at
                  399        nout_new =  0;
                  400        tout_new = [];
                  401        yout_new = [];
                  402        while next <= ntspan
                  403          if tdir * (tnew - tspan(next)) < 0
                  404            if haveEventFcn && stop     % output t
                  405              nout_new = nout_new + 1;
                  406              tout_new = [tout_new, tnew];
                  407              yout_new = [yout_new, ynew];
                  408            end
                  409            break;
                  410          end
```

```
polate.

;

.
gs,...
, nfailed, nfevals],...
.
, yeout, ieout,...
});
```

```
                411          nout_new = nout_new + 1;
                412          tout_new = [tout_new, tspan(next)];
                413          if tspan(next) == tnew
                414            yout_new = [yout_new, ynew];
                415          else
                416            yout_new = [yout_new, ntrp45(tspan(nex
                417          end
                418          next = next + 1;
                419        end
                420      end
                421
< 0.01    53018  422      if nout_new > 0
< 0.01    53018  423        if output_ty
< 0.01    53018  424          oldnout = nout;
< 0.01    53018  425          nout = nout + nout_new;
< 0.01    53018  426          if nout > length(tout)
< 0.01      614  427            tout = [tout, zeros(1,chunk,dataType)]
  0.01      614  428            yout = [yout, zeros(neq,chunk,dataType
< 0.01      614  429          end
  0.07    53018  430          idx = oldnout+1:nout;
  0.08    53018  431          tout(idx) = tout_new;
  0.05    53018  432          yout(:,idx) = yout_new;
< 0.01    53018  433        end
< 0.01    53018  434        if haveOutputFcn
                435          stop = feval(outputFcn,tout_new,yout_new
                436          if stop
                437            done = true;
                438          end
                439        end
< 0.01    53018  440      end
< 0.01    53018  441    end
                442
< 0.01    53018  443    if done
< 0.01     1088  444      break
                445    end
                446
                447    % If there were no failures compute a new h.
< 0.01    51930  448    if nofailed
                449      % Note that absh may shrink by 0.8, and that
  0.02    42786  450      temp = 1.25*(err/rtol)^pow;
< 0.01    42786  451      if temp > 0.2
```

```
< 0.01    42786   452          absh = absh / temp;
                  453        else
                  454          absh = 5.0*absh;
< 0.01    42786   455        end
< 0.01    42786   456      end
                  457
                  458      % Advance the integration one step.
< 0.01    51930   459      t = tnew;
< 0.01    51930   460      y = ynew;
< 0.01    51930   461      if normcontrol
                  462        normy = normynew;
                  463      end
< 0.01    51930   464      if NNreset_f7
                  465        % Used f7 for unperturbed solution to interp
                  466        % Now reset f7 to move along constraint.
                  467        f(:,7) = feval(odeFcn,tnew,ynew,odeArgs{:}),
                  468        nfevals = nfevals + 1;
                  469      end
  0.02    51930   470      f(:,1) = f(:,7);  % Already have f(tnew,ynew)
                  471
< 0.01    51930   472    end
                  473
  0.08     1088   474    solver_output = odefinalize(solver_name, sol,..
           1088   475                                outputFcn, outputArg
           1088   476                                printstats, [nsteps,
           1088   477                                nout, tout, yout,...
           1088   478                                haveEventFcn, teout,
           1088   479                                {f3d,idxNonNegative}
< 0.01     1088   480    if nargout > 0
< 0.01     1088   481      varargout = solver_output;
  0.03     1088   482    end
```

Other subfunctions in this file are not included in this listing.

gforce (Calls: 1138704, Time: 4.093 s)

Generated 09-Feb-2018 23:07:49 using performance time.
function in file D:\University of Colorado Academics\CU_S2018\ASEN 4057 - Aero Software\Assignments\Assignment 2\Final Code\gforce.m
Copy to new window for comparing multiple runs

Refresh

☑ Show parent functions ☑ Show busy lines ☑ Show child functions

☑ Show Code Analyzer results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
orbit_equations	function	1138704

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time
20	`D = euclidian_distance(Xa,Ya,X...`	1138704	2.342 s	57.2%
23	`Fx = G*Ma*Mb*(Xa-Xb)/(D^3);`	1138704	0.243 s	5.9%
26	`Fy = G*Ma*Mb*(Ya-Yb)/(D^3);`	1138704	0.225 s	5.5%
27	`end`	1138704	0.141 s	3.4%
17	`G = 6.674*10^(-11); %N.m^2/kg^...`	1138704	0.041 s	1.0%
All other lines			1.101 s	26.9%
Totals			4.093 s	100%

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time

Time Plot



Plot

euclidian_distance	function	1138704	1.229 s	30.0%	
Self time (built-ins, overhead, etc.)			2.865 s	70.0%	
Totals			4.093 s	100%	

Code Analyzer results
No Code Analyzer messages.

Coverage results
Show coverage for parent directory

Total lines in function	27
Non-code lines (comments, blank lines)	22
Code lines (lines that can run)	5
Code lines that did run	5
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

Function listing
Color highlight code according to time

```
time      Calls     line
                     1  function [ Fx,Fy ] = gforce( Xa,Ya,Xb,Yb,Ma,Mb )
                     2  % gforce() calculates the fore acting between tw
                     3  % distance, D, separating them. Assumes the newt
                     4  % constant.
                     5  %
                     6  % function call:
                     7  % [ Fx,Fy ] = gforce( Xa,Ya,Xb,Yb )
                     8  %
                     9  % inputs:
                    10  % Xa,Ya  = the x and y positions of the A body
                    11  % Xb,Yb  = the x and y positions of the B body
                    12  % Ma,Mb  = Masses of the bodies (kg)
                    13  %
                    14  % written 1/29/2018  Aaron Aboaf
                    15  % % % % % % % % % % % % % % % % % % % % % % % % %
                    16  % Define the gravitational constant
  0.04  1138704     17  G = 6.674*10^(-11); %N.m^2/kg^2
                    18
                    19  % Calculate the distance between the bodies base
```

)
wo bodies X and Y over a
tonian gravitational

(meters)
(meters)

% % % % % % % % % % % % %

ed on their coordinates

```
2.34  1138704   20  D = euclidian_distance(Xa,Ya,Xb,Yb);
                21
                22  % Calculate the force in the X direction
0.24  1138704   23  Fx = G*Ma*Mb*(Xa-Xb)/(D^3);
                24
                25  % Calculate the force in the Y direction
0.23  1138704   26  Fy = G*Ma*Mb*(Ya-Yb)/(D^3);
0.14  1138704   27  end
```

Other subfunctions in this file are not included in this listing.