MECH 222 Computer Lab 6:

Engine Optimization

Revised by Simon Tse in Mar 2014, using the version by Philip D. Loewen, Feb 2012.

Part I

Prelab Reading

IDEA 1 The Air-Standard Otto Cycle—Basic Theory

The Otto cycle is one way to transform heat energy into mechanical work. The following very rough illustration will help us describe it.

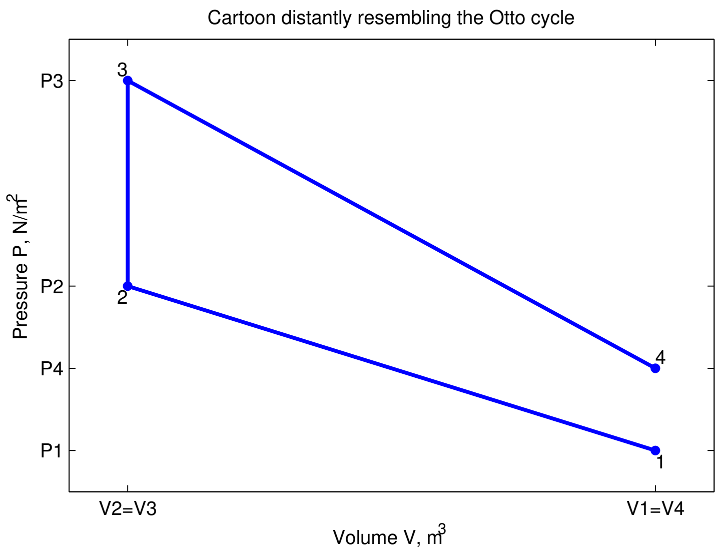


Figure 1: The Otto Cycle

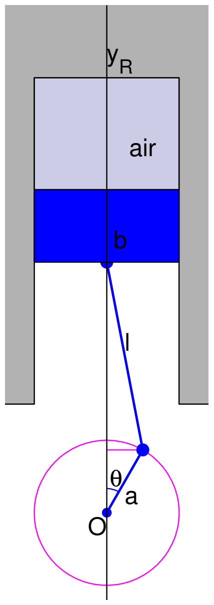


Figure 2: Piston Geometry

Fig. 1 suggests how pressure and volume evolve during the power stroke in one cylinder of a simplified internal-combustion engine. The cylinder is sketched in Fig. 2. In state 1, $\theta = -\pi$ and the piston is at "Bottom Dead Centre (BDC)"—the point where it leaves the maximum volume in the cylinder. (Call that volume V_1.) The cylinder is full of a fuel-air mixture. The piston in the cylinder compresses the gas leading to state 2. This is the compression stroke. In state 2, $\theta = 0$ and the piston is at "Top Dead Centre (TDC)"—the point where the volume in the cylinder (V_2) is smallest. Of course the corresponding pressure P_2 is larger than the initial pressure P_1. Then the spark plug fires and a rapid chemical reaction transforms energy stored in chemical bonds into kinetic energy of the gas molecules. The state jumps to position 3, where V_3 is the same as V_2 (the piston is still at TDC), but P_3 is much higher than P_2. Then the high pressure pushes on the piston as it moves back out to its original position (BDC), now described with crank angle $\theta = \pi$. This is the power stroke. It brings the system to state 4, where the enclosed volume V_4 is the same as the initial volume V_1. Of course the pressure in the cylinder, P_4, is higher than the initial pressure P_1 because the gas has higher temperature. To get back to state 1, ready to repeat the cycle, real engines make a whole extra revolution to expel the combustion products from the cylinder and replace them with a fresh fuel-air mixture (the exhaust and intake strokes). You can find a more detailed description on Wikipedia (en.wikipedia.org/wiki/Four-stroke_engine), and a nice animated diagram at http://www.animatedengines.com/otto.shtml.

Air-Standard Assumption and Other Idealizations

This lab deals only with one revolution of the crankshaft: the compression and power strokes, so the transition sequence $1 \to 2 \to 3 \to 4$ highlighted in Fig. 1 will be our exclusive focus. Combustion is too complicated to deal with in detail, so we assume the cylinder actually contains nothing but air ($k = 1.4$ in equation (3) and thereafter), and model the transition $2 \to 3$ as the instantaneous addition of some known amount of heat energy to the air by some unspecified process. *These simplifications characterize the "air-standard" model.* To get started, we will assume further that the cycle starts at standard atmospheric pressure $P_1 = 101$ kPa and temperature $T_1 = 333$ K. The mass of air in the cylinder, say m, stays the same throughout the process, so the following pressure-volume-temperature relationship is always in force:

$$PV = mRT. \qquad (1) \tag{1}$$

where R is the gas constant. If there is no heat transfer between the gas and the cylinder walls or the piston, the compression $(1 \to 2)$ and expansion $(3 \to 4)$ processes can be taken as adiabatic and reversible.

Thermodynamics

Two fundamental equations and one definition govern this process. The equations are the ideal gas law (1) and the First Law of Thermodynamics (simplified):

$$\delta Q = P\,dV + mC_v\,dT. \tag{2}$$

We define $k = 1 + R/C_v$, where R is the gas constant and C_v is the specific heat of the gas we are using. This leads to a convenient form of the First Law where C_v does not appear explicitly:

$$\delta Q = P\,dV + \frac{mR}{k-1}\,dT, \qquad \text{i.e.,} \qquad dT = \frac{k-1}{mR}\left[\delta Q - P\,dV\right]. \tag{3}$$

In our scenario, the mass m in the cylinder is constant. Thus differentiating the gas law (1) using product rule and dividing through by the original equation would give

$$\frac{dP}{P} + \frac{dV}{V} = \frac{dT}{T}.$$

We can replace the right side here using the First Law in form (3):

$$\frac{dP}{P} + \frac{dV}{V} = \frac{k-1}{mRT}\left[\delta Q - P\,dV\right]. \tag{4}$$

Here the combination mRT on the right side equals PV by (1), so we have the following important relationship between pressure, volume, and heat content:

$$\boxed{\frac{dP}{P} = \frac{k-1}{PV}\,\delta Q - \frac{k}{V}\,dV.} \tag{5}$$

Notice that this formula has only one constant parameter k, which coming from physical measurements of the gas.

In an adiabatic, reversible process, $\delta Q = 0$ and (5) reduces to a differential equation describing a famous relationship between P and V:

$$\frac{1}{P}\frac{dP}{dV} = -\frac{k}{V}. \tag{6}$$

In Fig. 1, the transition from state 1 to state 2 should follow a path compatible with (6); likewise for the transition from state 3 to state 4. The true paths are *not straight lines*!

In any reversible piston-moving scenario, the mechanical work done by the gas on the outside world is

$$W = \int dW = \int P\,dV. \tag{7}$$

This is a "line integral", so orientation matters. In the transition from state 1 to state 2, $dV < 0$ leads to a negative result: this makes sense because it's the compression stroke, in which the gas is receiving energy from outside. By contrast, in the transition from state 3 to state 4, $dV > 0$ and the contribution to W is positive because the gas is expanding and sending mechanical energy to the outside. (In the idealized case of Fig. 1, where state 2 and state 3 have the same volume, we have $dV = 0$ for the segment joining them. Since the piston doesn't move, no work goes either way.)

The efficiency of heat-to-work conversion along the path $1 \to 2 \to 3 \to 4$ is the ratio of work output to heat input (generated by the combustion of fuel):

$$\eta = \frac{W}{\bar{Q}},$$

where W is the work in line (7) and $\bar{Q} = \int \delta Q$ is the line integral accounting for heat addition. Taken together, we can express efficiency as a ratio of line integrals:

$$\eta = \frac{W}{\bar{Q}} = \frac{\int P\,dV}{\int \delta Q}. \tag{8}$$

IDEA 2 Crank Angle Parametrizes Everything

As the piston makes one cycle through compression, spark, and expansion, the crankshaft at O keeps turning. Thus every point on the state-transition path $1 \to 2 \to 3 \to 4$ in Figure 1 is associated with a specific crank angle θ in the mechanism shown in Figure 2. We can use this variable to parametrize all the paths we need to analyze.

The volume in the cylinder as a function of crank angle θ can be found using standard geometric reasoning (see the Appendix): it is

$$\frac{V(\theta)}{V_d} = \frac{1}{r-1} + \frac{1}{2}\left[1 + R_1 - \cos\theta - \sqrt{R_1^2 - \sin^2\theta}\right] \tag{9}$$

Here V_d is the piston's displacement, R_1 is a dimensionless ratio of link lengths in the piston mechanism, and r is the dimensionless ratio of maximum to minimum volume in the cylinder. Differentiating with respect to θ gives

$$\frac{V'(\theta)}{V_d} = \frac{1}{2}\sin\theta\left[1 + \frac{\cos\theta}{\sqrt{R_1^2 - \sin^2\theta}}\right] \tag{10}$$

Angle $\theta = 0$ gives the minimum gas volume, $V(0) = V_d/(r-1)$.

Heat addition is *fast*, but it can't be truly *instantaneous*. So the constant-volume transition between states 2 and 3 shown in Figure 1 is one of the model's more aggressive simplifications. Experts suggest that heat generated by combustion might actually depend on crank angle like this:

$$Q(\theta) = \int_0^\theta \delta Q = \begin{cases} 0, & \text{if } -\pi \le \theta \le \theta_s \\ \dfrac{\bar{Q}}{2}\left(1 - \cos\left(\dfrac{\pi(\theta - \theta_s)}{\theta_b}\right)\right), & \text{if } \theta_s < \theta < \theta_s + \theta_b \\ \bar{Q}, & \text{if } \theta_s + \theta_b \le \theta \le \pi \end{cases} \tag{11}$$

Here we can see the constant value $Q = 0$ before the "spark angle" θ_s and the higher constant value $Q = \bar{Q}$ for all angles after $\theta_s + \theta_b$, signifying completion of combustion. Here θ_b is the "burn angle"—the amount the crank turns during heat addition; it is typically quite small, but not zero.

These features are also visible in the derivative formula

$$Q'(\theta) = \begin{cases} 0, & \text{if } -\pi \le \theta \le \theta_s \\ \dfrac{\pi \bar{Q}}{2\theta_b}\left(\sin\left(\dfrac{\pi(\theta - \theta_s)}{\theta_b}\right)\right), & \text{if } \theta_s < \theta < \theta_s + \theta_b \qquad (12) \\ 0, & \text{if } \theta_s + \theta_b \le \theta \le \pi \end{cases} \tag{12}$$

Here are figures showing the functions Q and Q' for certain values of θ_s and θ_b; we take $\bar{Q} = 1800$ J.

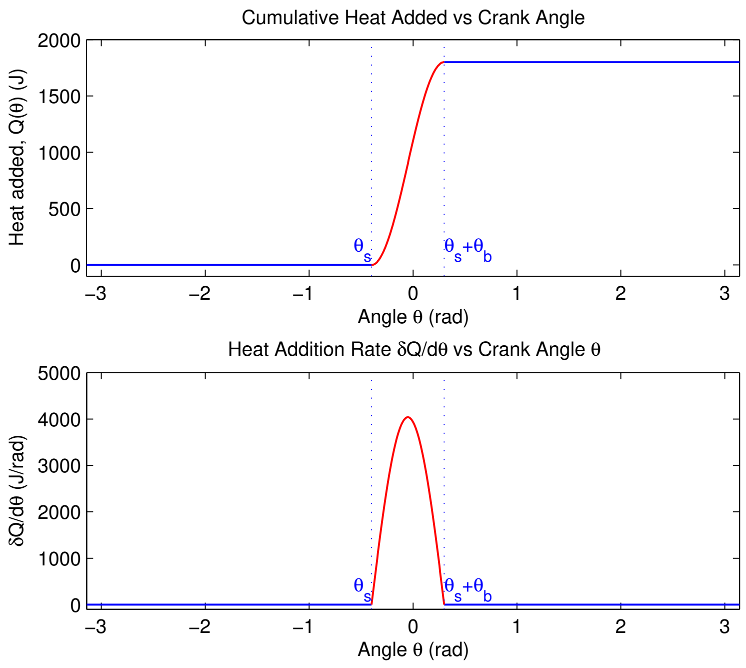


Figure 3: Heat input rate and total heat added, versus crank angle

The spark angle θ_s is an adjustable feature of most standard internal-combustion engines; the burn angle θ_b can be influenced by chemical properties of the fuel-air mixture and by mechanical considerations outside the scope of our work here. These two angles are important design parameters for engine-makers.

Dividing the boxed equation (5) above by $d\theta$ produces a differential equation that reveals the θ-dependence of the pressure $P = P(\theta)$:

$$\boxed{\frac{dP}{d\theta} = \frac{k-1}{V(\theta)}\,Q'(\theta) - \frac{kP(\theta)}{V(\theta)}\,V'(\theta).} \tag{13}$$

We already have detailed formulas for $V(\theta)$, $V'(\theta)$, and $Q'(\theta)$, and we know the initial condition $P(-\pi) = P_1$. So we can use (6) to find $P(\theta)$ and draw all sorts of interesting conclusions. We will use Matlab's built-in ODE solver `ode45` to do this job.

Summary

Using the crank angle θ as a parameter, the interval $-\pi < \theta < \pi$ captures all four thermodynamic states in Figure 1. We have $\theta = -\pi$ in state 1, $\theta \approx 0$ for states 2–3 (often a poor approximation), and $\theta = \pi$ in state 4. We get $V(\theta)$ from geometry (line 9), $Q(\theta)$ by inventing something plausible (line 11), and $P(\theta)$ by solving a differential equation (line (13)). Once all three functions are known, we can calculate the work done by the piston as

$$W = \int P\,dV = \int_{\theta=-\pi}^{\pi} P(\theta)V'(\theta)\,d\theta \tag{14}$$

and deduce the cycle's thermodynamic efficiency from line 8 to give the readily computable formula:

$$\eta = \frac{W}{\bar{Q}} = \frac{\int_{\theta=-\pi}^{\pi} P(\theta)V'(\theta)\,d\theta}{\bar{Q}}. \tag{15}$$

IDEA 3 Global Variables and the Struct Data Type

In this lab we will write many functions that uses the same set of physical quantities. Instead of passing them explicitly as function input arguments, we will save their values in *a global variable*. Indeed, there will not be many such variables, but just one variable of struct type, which conveniently groups and contains all these values as its fields.

This *global struct* variable is named `GLOBAL_ENGINE_PAR`, which quite literally means the global engine parameters, to be initialized with default values by the script `cyclesetup.m`, so that you will not need to risk typing them wrong.

To access the values stored in the fields of `GLOBAL_ENGINE_PAR`, we use a dot "`.`" between the variable name and field name:

```
GLOBAL_ENGINE_PAR.V_D = 5.E-4; % Piston displacement (m^3)
```

is a line that assigns the Piston displacement volume to the field `V_D`. When we use these values, we need to first declare the variable to be global to obtain access. For instance, in the provided function m-file `qdq.m`, there are the lines:

```
global GLOBAL_ENGINE_PAR % To get access the GLOBAL_ENGINE_PAR.
V_D = GLOBAL_ENGINE_PAR.V_D;
```

which gets access to the default values outside of the function workspace and then saves a *local* copy with a more readable and conveniently short name.

Global variables are used precisely because they are convenient. However, convenience comes at the expense of increased risk, because the more they are accessed (for convenience), the more places a wrong value can screw things up (to create intractable bugs that may possibly reside in every file that can access this global variable). Usually we exercise caution by giving them very distinctive names, and only read from it most of the times. However, they can actually be modified whenever they can be accessed. Here's some of what Matlab has to say in response to "`doc global`":

> "`global X Y Z` defines `X`, `Y`, and `Z` as global in scope.
> Ordinarily, each MATLAB function, defined by an M-file, has its own *local variables*, which are separate from those of other functions, and from those of the *base workspace.* However, if several functions, and possibly the base workspace, all declare a particular name as global, they all share a single copy of that variable. Any assignment to that variable, in any function, is available to all the functions declaring it global. "

As a pedagogical exercise, we will actually modify some fields of `GLOBAL_ENGINE_PAR` in the last lab activity to understand how global variable dispatch new values without passing them as function arguments. In the quoted Matlab documentation, the mentioned term *workspace* is an important concept to master should you intend to do large projects in Matlab, so does learning the more flexible data types like *cell* and *struct.* There is an entire section in the Appendix for motivated readers.

PRE-LAB EXERCISES

All four questions below refer to the states and transitions shown in Figure 1 above.

PL1: For a reversible transition of an ideal gas in the piston mechanism, the evolution of pressure and volume in time obeys

$$\frac{dP}{P} = \frac{k-1}{PV}\,\delta Q - \frac{k}{V}\,dV. \qquad (*)$$

In addition, when the gas expansion is adiabatic, i.e. $\delta Q = 0$ during the transition, (*) becomes a separable ODE that can be solved explicitly.

Solve the ODE to express $P(\theta)$ in terms of $V(\theta)$ and the initial values $P_0 = P(\theta_0)$ and $V_0 = V(\theta_0)$ where θ_0 is the starting crank angle.

PL2: Assume that in state 1, the gas pressure, volume, and temperature are

$$P_1 = 1.01 \times 10^5 \mathrm{N/m}^2, \qquad V_1 = 5.556 \times 10^{-4} \mathrm{m}^3, \qquad T_1 = 333\mathrm{K}.$$

and that the transition $1 \to 2$ is reversible and adiabatic. If the piston compresses the gas volume to $V_2 = 0.556 \times 10^{-4}$ m^3, find the corresponding values of P_2 and T_2 using PL1.

PL3: Assume that during the transition $2 \to 3$, combustion of fuel results in heat addition of $\bar{Q} = 1800J$, and that the volume of fuel-air mixture is kept fixed. Then, the equation $(*)$ holds with $dV \equiv 0$, so $V \equiv V_2 = V_3$, and integration gives

$$(P_3 - P_2) = \frac{k-1}{V}\bar{Q}.$$

Calculate P_3 and T_3.

PL4: Again assume adiabatic and reversible expansion during the transition $3 \to 4$ where the volume of gas returns to the initial state, so $V_4 = V_1$. Calculate P_4 and T_4.

You will find it handy for the lab if you type in the values of k, $\bar{Q}$ and P_i, V_i, T_i ($i = 1, 2, 3, 4$) and save the formula you find in PL1. The lab assignment will begin by plotting these transitions.

Part II

Lab Assignment

ACTIVITY 1 PV-trajectories for the ideal Otto cycle.

Figure 1 in the Prelab reading is just a conceptual sketch. We now use the answers from Prelab questions to produce more accurate plots. In this activity, take the same values for the initial state and heat addition as in the Prelab, which we reproduce below:

$$P_1 = 1.01 \times 10^5 \mathrm{N/m}^2, \qquad V_1 = 5.556 \times 10^{-4} \mathrm{m}^3, \qquad T_1 = 333\mathrm{K}, \qquad \bar{Q} = 1800J$$

Use `subplot` to generate a horizontal 2-pane figure:

1.1 Left figure: show transitions $1 \to 2 \to 3 \to 4$ on axes where absolute pressure P in $\mathrm{N/m}^2$ is plotted against volume V in m^3. (Same axes as in Figure 1.) Use the curves whose equations you derived in PL1 for $1 \to 2$ and $3 \to 4$, and a straight vertical line for the transition "$2 \to 3$".

1.2 Right figure: plot base-10 log of the pressure ratio P/P_1 against the base-10 log of the volume ratio V/V_1.

Save the 2-pane figure as `idealotto.fig;` sample plots are given below:

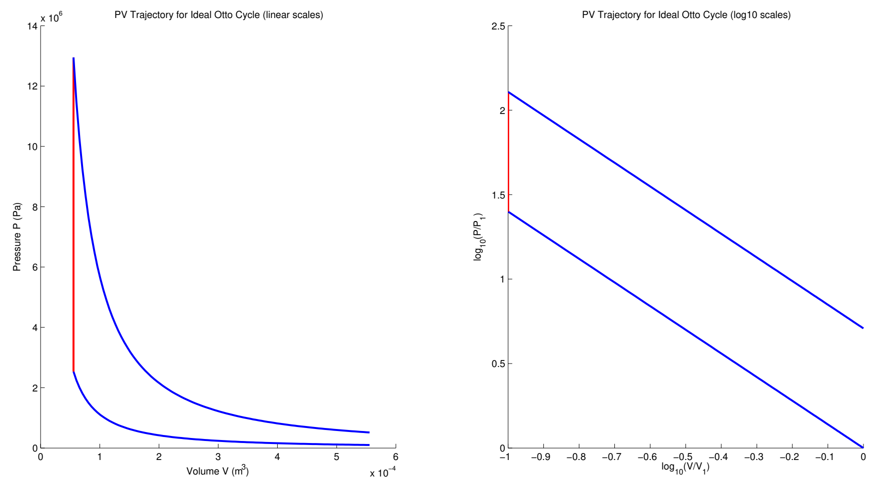


Figure 4: The ideal Otto cycle

Hand-in Checklist:

- `idealotto.fig`, something similar to Fig. 4.

ACTIVITY 2 Visualize volume and energy changes in each cycle.

2.1. Locate and save three m-files `cyclesetup.m`, `vdv.m` and `qdq.m` into your current working folder. Read and run script `cyclesetup` which initializes a global struct variable `GLOBAL_ENGINE_PAR` to contain the physical constants for the engines:

$$k = 1.40, \qquad \bar{Q} = 1800J, \qquad P_{amb} = 1.01 \times 10^5 \mathrm{N/m}^2, \qquad V_d = 5.00 \times 10^{-4}, \qquad r = 10.0$$

and also the spark and burn angles we would like to tune later on (which currently have unrealistic values):

$$\theta_s = 0, \qquad \theta_b = 0$$

Read also the code for functions

```
[V,dV]=vdv(theta); [Q,dQ]=qdq(theta);
```

which are vectorized and implement formulas (9) through (12) for V, V' and Q, Q' respectively. Observe how they connect to the global workspace, and access the fields of a struct variable to obtain the values they need.

In this and all subsequent activities (and also throughout the Prelab reading), all θ-dependent plot and calculations should assume this interval:

$$-\pi < \theta < \pi$$

Draw two-pane horizontal plots for each of the following items.

2.2. Illustrate gas volume versus crank angle and save it as `vdvplot.fig`. Put the volume V in the top pane, and the rate of change $V'(\theta) = dV/d\theta$ in the bottom.

2.3. Illustrate heat addition versus crank angle and save it as `qdqplot.fig`. Plot the cumulative heat addtion $Q(\theta)$ vs θ in the top pane, and show the rate of change with respect to the crank angle, $Q'(\theta)$ vs θ, in the bottom. Use default values stored in the fields of `GLOBAL_ENGINE_PAR`, but change the spark angle θ_s and the burn duration angle θ_b according to your lab day as tabulated below (Note: The numbers given above are in *degrees*, but all computations in the Prelab reading and all provided code assumes *radians*):

θ_s (degrees) by lab day ...	Mon:	− 20,	Tue:	− 15,	Wed:	20,	Fri:	− 40;
θ_b (degrees) by lab day ...	Mon:	60,	Tue:	45,	Wed:	40,	Fri:	30.

Your plot should look like Figure 3 on page 4.

Hand-in Checklist:

- `vdvplot.fig` and `qdqplot.fig`, two double-paned figures that visualizes the crucial formulas (9) through (12) for V, V',Q, Q' as functions of θ.

ACTIVITY 3 Simulate non-instantaneous heat generation

Unlike the ideal case assumed in the Prelab exercises and Activity 1, heat generation depends on time (and thus on crank angle) and is *not* instantaneous. The plots from Activity 2 gives us the gas's volume V and added heat Q as functions of the crank angle θ (which increases with time). Then, as explained by the Prelab writeup, the pressure P must satisfy the ODE (13), which we reproduce below:

$$P'(\theta) = \frac{k-1}{V(\theta)}\,Q'(\theta) - \frac{kP(\theta)}{V(\theta)}\,V'(\theta), \qquad P(-\pi) = P_{amb}. \qquad (*)$$

The initial condition assumes that the initial pressure P_1 equals the ambient pressure P_{amb}. We will use `ode45` to solve this initial-value problem for the function P.

3.1 The right hand side of the equation $(*)$ is provided in the file `dpdtheta.m` from the assignment page. Read the content of `dpdtheta.m`, and find out how many and which fields of the global struct `GLOBAL_ENGINE_PAR` would be accessed when a command such as `dpdtheta(0,1.01E5)` is run. Put your answer to this question as comments near your function call of `dpdtheta` in the new function `pdp` to be written in the next item.

3.2 Write a Matlab function `pdp` starting with the line:

```
[P,dP] = pdp(theta)
```

using `ode45` to solve for $P = P(\theta)$ in $(*)$. The argument `theta` is going to be a vector representing a high resolution grid on $-\pi < \theta < \pi$. The following are essential elements in your program:

- a line that declares the struct variable `GLOBAL_ENGINE_PAR` to be global and create local copies of the necessary constants for $(*)$ from its fields; this practice has been demonstrated in the provided functions `qdq` and `pdp`
- a call to `ode45` containing the 3 essential ingredients of an initial value problem:
  - function handle for the right hand side of $(*)$, check whether it takes the form `du=f(t,u)` assumed by `ode45`.
  - a vector of nodes `tspan` for the time-like independent variable `t`, so that `u(tspan(i))` is computed and returned for each `i`
  - an initial state `u0`

  All the ingredients should already be available to you – your work is to identify them and put them in the right order and syntax.
- you do not need to (re)name them exactly as `f, t, u, tspan, u0` etc., it is more natural to use the original letters that defines the IVP
- request a smaller relative error tolerance by declaring a struct variable `odeopt` with a single field:

  ```
  odeopt.Reltol = 1.E-6; % Default val 1.E-4 is sloppy
  ```

  and pass `odeopt` as the fourth argument to `ode45`. This is a quick alternative to using the Matlab function `odeset`.

3.3 Load your figure from ACTIVITY 1, then use your function `pdp` to find values for $P(\theta)$ for the whole cycle $1 \to 2 \to 3 \to 4$. Overlay the computed PV-trajectories on the previous ones you produced in ACTIVITY 1 in both plots. Improve clarity by using different colors and/or line styles to distinguish the two curves.

Save it as `realotto.fig`. The resultant figure should look like:

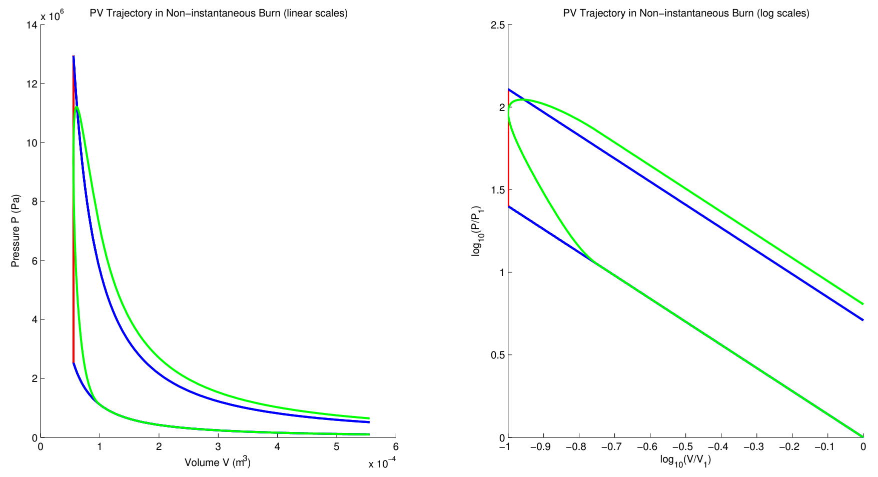


Figure 5: A more realistic Otto cycle

Hand-in Checklist:

- `pdp.m`, a function m-file that imitates the structure of `vdp.m` and `qdq.m`, but solves an ODE to compute $P(\theta)$ instead of using explicit formulas.
- `realotto.fig`, a Matlab figure like Fig. 4 above.

ACTIVITY 4 Optimize thermodynamic efficiency.

Let us tune our engine to an optimal efficiency.

4.1. Write a function `enginetune` that calculates the mechanical work done W and the engine efficiency η which uses the function inputs of θ_s and θ_b to temporarily override the global values.

A template of `enginetune.m` is provided. Follow the instructions below to compute the two return values:

- `W`: the mechanical work done W along the $PV-$trajectory plotted above. The relevant formula is reproduced here:
$$W = \int P\,dV = \int_{\theta=-\pi}^{\pi} P(\theta)V'(\theta)\,d\theta.$$
Trapezoidal rule is a reasonable method to use – i.e. use Matlab `trapz` to find a good approximation of W.
- `eta`: the efficiency is the fraction that compare the mechanical work generated compared to the heat energy input:
$$\eta = W/\bar{Q}$$

WARNING: This function `enginetune` is not vectorized, i.e. the inputs `theta_s` and `theta_b` should be single values. A call to `qdq` will fail if the fields `THETA_S` and `THETA_B` are not single-valued.

Now keep the burn duration angle θ_b as specified for your lab day in item 2.5 but step the spark angle θ_s to find the optimal value:

4.2 Step the spark angle θ_s through 91 equally-spaced choices from $-\theta_b$ to θ_b, inclusive. For each choice of θ_s, find its thermodynamic efficiency η, and then find the efficiency ratio $\eta/\eta_{\max}$ where the theoretical ideal value $\eta_{\max}$ is given by
$$\eta_{\max} = 1 - \frac{1}{r^{k-1}}.$$

Give 2 plots to illustrate your findings:

- Plot the efficiency ratio $\eta/\eta_{\max}$ versus angle θ_s as a collection of dots. Identify the maximum efficiency value the θ_s that achieves it (among the 91 choices). Save the plot as `sparkoptim.fig`.
- Again overlay the optimal PV-trajectory on the ideal ones, as done in item 3.3 (it should now be closer to the ideal one). Save the plot a `optimotto.fig`.

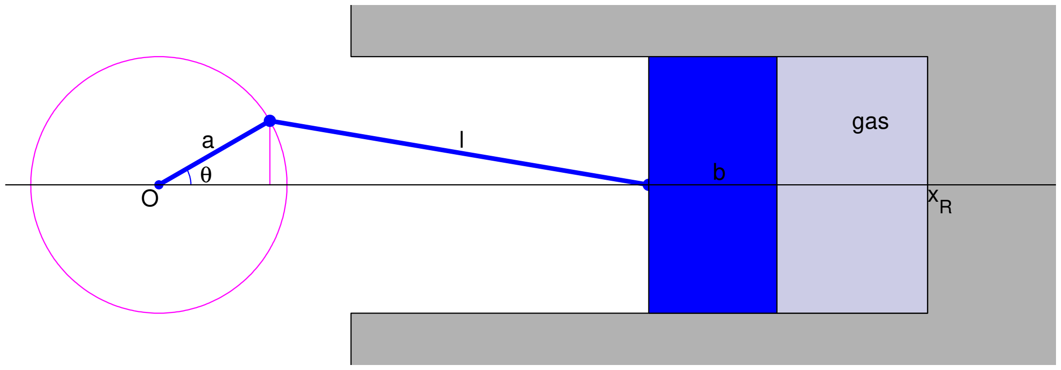
Save all your commands in the script `optimengine.m`. Ensure that it can be run from a cleared workspace (so it should call `cyclesetup.m` in the beginning). It will be nice if you format the script comments in a way that is publishable by Matlab `publish`, but this is not required.

Hand-in Checklist:

- `enginetune.m`, the function which overrides some global values to find mechanical work done and efficiency.
- `optimengine.m`, a script which optimizes the spark angle and present the result in two plots.
- `sparkoptim.fig` and `optimotto.fig`, the two plots created by running `optimengine`

Appendix: Piston Geometry

Consider the piston/crank assembly in Figure 2 of the main writeup. The crank pivots at the origin.



For the angle θ shown here, the x-coordinate of the piston's face is

$$x(\theta) = a\cos\theta + \sqrt{\ell^2 - a^2\sin^2\theta} + b.$$

The extreme positions of the piston's active face are clearly

$$x_{\text{MAX}} = x(0) = a + \ell + b,$$
$$x_{\text{MIN}} = x(\pi) = -a + \ell + b.$$

Here a is the length of the crank arm, and the piston's "stroke" is $2a$. If A denotes the area of the piston face, then the volume of the region the piston face moves through in each cycle, called the "displacement", is $V_d = 2aA$. Suppose the top end of the piston chamber is at location x_R. Then the extreme volumes inside the chamber will be

$$V_0 \triangleq V_{\text{MIN}} = A(x_R - x_{\text{MAX}}) = A(x_R - b - a - \ell), \qquad (*)$$
$$V_{\text{MAX}} = A(x_R - x_{\text{MIN}}) = A(x_R - b + a - \ell).$$

The standard name for V_{MIN} is the "clearance volume"; the standard symbol is V_0. Hence

$$V_{\text{MAX}} = V_0 + V_d.$$

Piston experts work with two dimensionless ratios:

$$R = \frac{\ell}{a}, \qquad r = \frac{V_{\text{MAX}}}{V_{\text{MIN}}} = \frac{V_0 + V_d}{V_0}.$$

Note that $r - 1 = V_d/V_0$. The volume ratio between enclosed volume and piston displacement is

$$\frac{V(\theta)}{V_d} = \frac{A(x_R - x(\theta))}{A(2a)} = \frac{1}{2a}\left[x_R - b - a\cos\theta - \sqrt{\ell^2 - a^2\sin\theta}\right]$$
$$= \frac{1}{2}\left[\frac{x_R - b}{a} - \cos\theta - \sqrt{R^2 - \sin^2\theta}\right]$$

Solving for $x_R - b$ in $(*)$ gives

$$x_R - b = \frac{V_0}{A} + a + \ell = 2a\frac{V_0}{V_d} + a(1 + R).$$

Back-substitution yields

$$\frac{V(\theta)}{V_d} = \frac{1}{2}\left[2\frac{V_0}{V_d} + 1 + R - \cos\theta - \sqrt{R^2 - \sin^2\theta}\right] = \frac{1}{r-1} + \frac{1}{2}\left[1 + R - \cos\theta - \sqrt{R^2 - \sin^2\theta}\right].$$

In the main lab writeup, we write R_1 instead of R for the length ratio, to avoid a notational collision with the ideal gas constant.

Reference http://www.engr.colostate.edu/~allan/thermo/page2/page2.html

Appendix: Matlab Facilities for a More Complicated Project

As the complexity of your program increases beyond what you can handle within one programming session, or in other words, the number of variables, functions and m-files have grown to a size that you cannot remember without reviewing your codes from top to bottom, and perhaps again. In such a situation, you will find that certain good practices including extensive commenting, version control and most importantly, the *readability* of your code to be of paramount importance.

For readability of the code per se, using appropriate data types or just naming your variables in a more systematic way will already help tremendously to clarify your program logic and keep your code from getting unnecessarily long. We will discuss a scenario where we group variables of similar purpose under one umbrella in the subsection ***Data Types.***

Another way to improve readability, especially when your program have many variables and functions, which also require many inputs, then obtaining values of the variables and passing their names around become an important design issue. Worse still, if names do repeat in different usage scenarios, it is important to design a way to distinguish them and confine the scope of the name in their supposed usage scenario. For example, we use `i` or `n` everyday as a temporary variable for iteration, what guarantees that my for-loop will not pick up side effects from a function call that also contains a for-loop? In Matlab, the solution is to segregate variables by the boundary of workspaces, to be discussed in the subsection ***Workspaces.***

Data Types

All along the labs in this term, you have already picked up how to use the two most ubiquitous data type in Matlab: double and character arrays, either by means of examples or practice: any vector or matrix is an array of double precision floating point numbers and any string bounded by two single quotes is a character array.

For ideas like a list of coordinates and a certain message to be printed to the screen, double and character arrays are convenient enough. However, as we have seen in the report template for Computer Lab 5, when we want to save a table that contains both names of variables and their values (just like what you would do in a spreadsheet), neither double nor string array can do the job satisfactorily, because they are *pure* arrays. Matlab has two major heterogeneous data types: *cell* and *struct.* We will not discuss cell arrays in detail here except that the value each cell holds can be of *arbitrary type*: numeric, character, function_handle or even cell or struct.

Hierarchical data structure

Your data may relate to each other in a hierarchical relationship. The simplest example is that there is a group of variables with a same purpose or that they pertain to a common object. For example, in this lab, there are many parameters involved in the formulation of the Otto cycle, some are physical constants, some are particular measurements of the engine design, for instance. For clarity and convenience, it will be nice if we can group them in an intuitive way for later access. The *struct* data type comes up naturally in such a scenario.

A *struct* variable contains *fields* instead of a single value, and each field can contain a value of any type. Actually, we have seen an example in Computer Lab 6 when we adjusted the absolute and relative tolerances for `ode45`, perhaps without noticing or understanding how it works. The fact is that the ODE options-controlling variable `odeopt` is a *struct* with many fields. When we only want to change two of the fields, then the following

```
odeopt.Reltol = 1e-6;
odeopt.Abstol = [1e-6 1e-6];
```

is an alternative way to create the struct variable `odeopt` for `ode45` used in the hill-climbing simulation. Another toy example would be:

```
pointmass.x = [0 2 .3];
pointmass.y = [1 -1/2 2];
pointmass.z = [2 1 1];
pointmass.m = [3 4 2];
pointmass.name = 'black holes';
```

This *struct* variable has the name `pointmass`, and five fields `x`, `y`, `z`,m and `name`: four *double* arrays (Matlab assumes numbers to be double precision by default) and one *character* array. The fields of a struct are accessed by a dot "`.`" after the struct name, before the field name. Calling the name of the struct `pointmass` without suppressing the output gives the following:

```
pointmass =
        x: [0 2 0.3000]
        y: [1 -0.5000 2]
        z: [2 1 1]
        m: [3 4 2]
        name: 'black holes'
```

In this lab, we try to gather engine parameters with generic mathematical variable names into the fields of a struct variable `GLOBAL_ENGINE_PAR`. This ensures that the user knows that, for instance, the ambient pressure `PAMB`, piston volume `V_D` and spark and burn angles `THETA_S` and `THETA_B` *pertains* to an engine prototype. This also reduces the risk of repeating a generic name like `THETA_S`, that probably is also used in another model, say, a pendulum.

Workspaces: Base, Local and Global

Variable must belong to a workspace in Matlab, and it is vital that variable names be *unique* in each workspace, and mixing up names from different workspaces is a source of many bugs.

When you run a Matlab script, all the variables are accessed from and assigned to the ***base*** workspace, as if you were typing the commands in the command window line by line. Therefore, having too many scripts very often result in either an inadvertent override of previous values for a different script to work, or an uncleared value or setting or plot making some surprising impact on your current script. That is also why too many calls to `clear all` and `close all` signals a bad program design.

In place of overusing `clear all`, creating a function automatically alleviates the headache of unexpected variable name intrusion, because you define explicitly what input values are expected. In terms of workspace, each function, when called, creates a separate ***local*** workspace with private variables, not accessible from other functions or the command line.

Here is when the conundrum arises: what if we have a collection of functions that require many common inputs? Writing many scripts instead is said to be dangerous and cumbersome, but declaring a long list of input does not seem elegant either. Very often, programmers resort to a convenient middle ground: *global variables.*

In Matlab, global variables have a special design in that they occupy another *separate* workspace, and accessing the global workspace from a script (or command line) and a function requires an explicit declaration as early as possible. For instance, in this lab, we use this line

```
global GLOBAL_ENGINE_PAR
```

which *connects* a variable named `GLOBAL_ENGINE_PAR` in the base or local function workspace to the global workspace. The name is deliberately capitalized to highlight its nature as a global variable. Some other programming languages use idiosyncrasies including prefixing by underscores. Such precaution is needed because the more it is used, the more hazard it can create.

Matlab does not force you to exercise a certain strategy when using global variables. However, if you work in a tebam, or alone in a large project, using global variables without any measures to prevent it from being misused is going to cost you dearly. Practise *defensive* programming against human errors, primarily your own.

Further Reading

At your stage, the best place for further reading on using Matlab might be a source of good code examples. You can find many of these in Matlab example files. For instace, there are many example files for the `ode45` solver, where you see how the design of the `ode` functions are used in the intended way. However, the Matlab documentation is unfortunately more like an API with slightly more explanations, so its primary aim is to give sufficient details for the experts to exploit every feature. For beginners' learning, it is better to consult proven code written with extensive comments. Matlab claims there are 1000+ books written about Matlab, so perhaps we have one in our library.