Билет 3

1. **Прямые методы решения вариационных задач.**

Наряду с задачами, в которых необходимо определить максимальные или минимальные значения некоторой целевой функции, при проектировании новых объектов нередко возникает необходимость нахождения функций, доставляющих экстремум целевому функционалу. Такие задачи имеют место, например, при проектировании трубчатых химических реакторов (здесь требуется найти функцию распределения температуры по длине реактора, максимизирующую производительность), и многих других объектов.

Для решения таких задач разработаны многочисленные аналитические и численные методы. Преимущество аналитических методов заключается в получении точного решения, недостаток - узкий класс задач, которые могут быть решены этими методами. Численными методами (особенно прямыми) могут быть решены многие задачи, не имеющие аналитического решения.

Немного теории:

Функционал – оператор, ставящий в соответствие некоторой функции число. Аргумент функционала - $x(t)$.

Простейшая вариационная задача ставится следующим образом:

найти функцию $x^*(t)$, доставляющую экстремум функционалу вида:

$$J = \int_{t_0}^{t_1} f(t,x,x')dt \to extr$$

При этом для функции x(t) должны выполняться краевые условия $x(t_0)=x_0, x(t_1)=x_1$.

Прямые методы решения вариационных задач.

Прямые методы заключаются в нахождении искомой функции, доставляющей экстремум функционалу, непосредственно ее подбором. При этом не используется необходимое условие экстремума функционала и не составляется уравнение Эйлера.

1. Метод Ритца.

В методе Ритца предложено искать решение среди линейных комбинаций заранее известных функций.

$$x = \sum_{i=0}^{n} a_i W_i(t) \qquad (**)$$

W_i - заранее известные заданные функции;

a_i - неизвестные коэффициенты.

После подстановки (**) в функционал подынтегральная функция представляет собой набор известных функций аргумента t с неизвестными коэффициентами. Интеграл может быть взят, в результате чего получим некоторую функцию $\varphi(a_0,a_1,...,a_n)$, для которой надо найти экстремум. Поскольку необходимо определить экстремум функции n переменных, то задача сводится от вариационной к конечномерной. Полученную задачу решают любым методом нелинейного программирования.

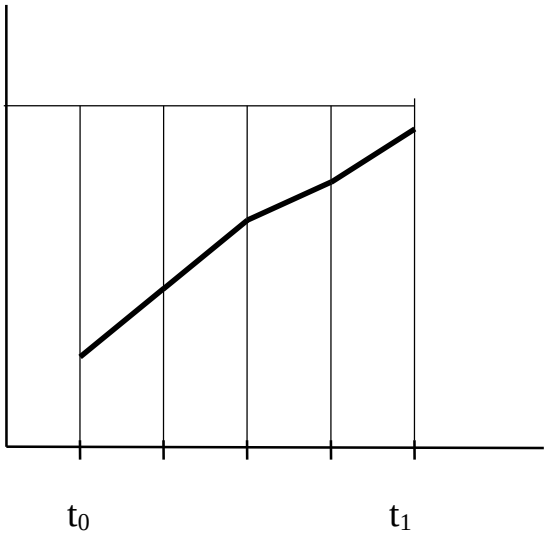
2. Метод Конторовича.

Имеет ту же основу что и метод Ритца, однако здесь допускается нелинейная относительно параметров a_i комбинация искомых функций.

Преимущества:

Может быть достигнута лучшая аппроксимация экстремали при меньшем количестве параметров a_i, в то же время более сложен вопрос о подборе функций удовлетворяющих краевым условиям.

3. Конечноразностный метод Эйлера.



Интервал $[t_0, t_1]$ разбивается на n частей. Задаются значений функции во внутренних точках отрезка. Вычисляется функционал численным методом. Меняются значения x_i любым методом нелинейного программирования, добиваясь экстремума функционала. С увеличением n решение стремится к точному.

2. Окна в компьютерной графике. Алгоритмы преобразования координат при выделении, отсечении элементов изображения.

Функции окна:

1. удаление (удаляется то, что в окне)
2. отсечение (остается то, что в окне)
3. выделение (цветом/толщиной)

Отсечение является одной из основных задач графики и применяется для того, чтобы предотвратить рисование тех частей объекта, которые размещаются вне заданной области. Рассмотрим классический алгоритм отсечения прямых — отсекатель Кохена—Сазерленда, который вычисляет, какая часть отрезка прямой с концевыми точками p1 и p2 лежит (и лежит ли вообще) внутри мирового окна, и возвращает концевые точки такой части отрезка. Существует много возможных расположений отрезка прямой по отношению к окну. Этот отрезок может находиться слева, справа, над и под окном; он может пересекать любую границу окна (или две) и т. д. Поэтому нам необходим систематизированный и эффективный метод, который идентифицировал бы существующую ситуацию и вычислял бы новые концевые точки для отсеченного отрезка.

Алгоритм Кохена—Сазерленда быстро выявляет и отбрасывает два распространенных случая, называемых «тривиальный прием» и «тривиальное отклонение». Как показано на рис. 3.15, обе концевые точки отрезка AB расположены

внутри окна W, и поэтому весь отрезок AB должен располагаться внутри W. Следовательно, отрезок AB может быть тривиально принят: он не нуждается в отсечении. Такая ситуация часто возникает при использовании большого окна, охватывающего большинство отрезков прямых. С другой стороны, обе концевые точки C и D лежат целиком по одну сторону от окна W, поэтому отрезок CD должен располагаться целиком вне W. Следовательно, CD тривиально отклоняется и не рисуется ничего. Такая ситуация часто возникает, когда маленькое окно используется для плотного изображения, многие из отрезков которого находятся вне этого окна.

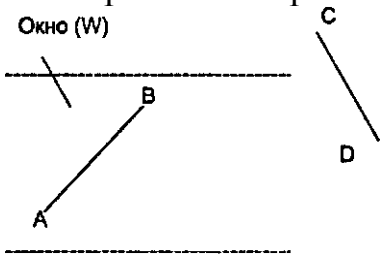


Рис. 3.15. Тривиальные прием или отклонение отрезка прямой

Проверка на тривиальный прием и тривиальное отклонение

Мы хотим быстро выявлять такие случаи, когда отрезок прямой может быть тривиально принят или тривиально отклонен. С целью облегчения этой задачи для каждой концевой точки отрезка вычисляется «кодовое слово внутри/вне». Рисунок 3.16 показывает, как происходит это вычисление. Точка P находится левее и выше окна W. Эти два обстоятельства записываются в кодовое слово для P. Буква T (TRUE) указывается для двух полей: «слева» от окна и «выше». Буква F (FALSE) указывается для двух остальных полей: «справа» от окна и «ниже».

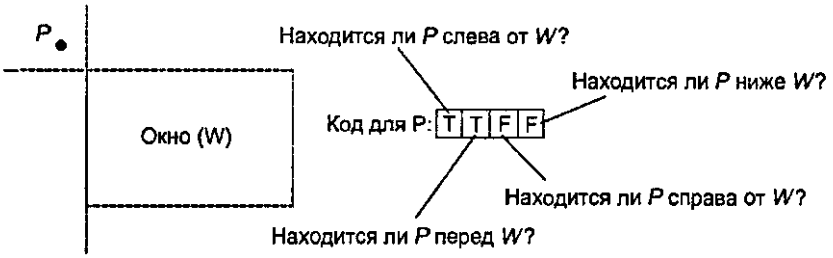


Рис. 3.16. Кодирование расположения точки P относительно окна

Пусть, например, точка P находится внутри окна, тогда ее код равен FFFF. Если P ниже, но ни слева, ни справа от окна, то ее код равен FFFT. На рис. 3.17 показаны все девять возможных расположений с кодом для каждого из них.

TTFF	FTFF	FTTF
TFFF	FFFF ОКНО	FFTF
TFFT	FFFT	FFTT

Рис, 3.17. Коды вне/внутри для точки

Мы формируем кодовое слово для каждой концевой точки тестируемого отрезка прямой. Условия тривиальных приема и отклонения легко связываются с этими кодовыми словами:

- тривиальный прием: оба кодовых слова равны FFFF;

•тривиальное отклонение: кодовые слова имеют Т в одном и том же месте, то есть обе точки находятся слева от окна или обе — справа и т. д.

Разделение отрезка в случае, когда не имеет места ни тривиальный прием, ни тривиальное отклонение.

Алгоритм Кохена— Сазерленда использует стратегию «разделяй и властвуй». Если отрезок не может быть ни тривиально принят, ни тривиально отклонен, он разделяется на две части по разные стороны одной из границ окна. Одна его часть располагается вне окна и отбрасывается. Вторая часть является потенциально видимой, поэтому весь процесс повторяется с оставшимся отрезком относительно других границ окна.

Алгоритм прекращает работу, пройдя данный цикл не более четырех раз, так как в каждой итерации мы оставляем только ту часть отрезка, которая «выдержала» тестирование относительно предыдущей границы окна, а таких границ всего четыре. Поэтому после максимум четырех итераций обеспечивается тривиальный прием или тривиальное отклонение.

Как осуществляется разделение на каждой границе? На рис. 3.18 показан пример, относящийся к правой границе окна.

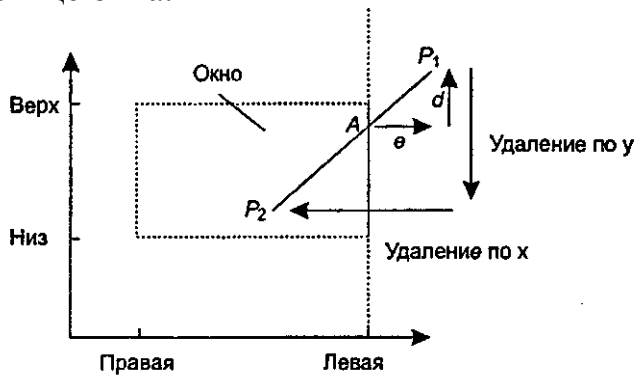


Рис. 3.18. Отсечение отрезка относительно границы

Положение точки А необходимо вычислить. Ее х-координата равна, очевидно, W. right - координате правой границы окна. Для определения ее у-координаты необходимо найти p1.y с помощью поправки d, как указано на рисунке. Однако из подобия треугольников

$$\frac{d}{\text{dely}} = \frac{e}{\text{delx}},$$

где e = pl.x – W.right, а равенства

```
delx = p2.x - p1.x;
dely = p2.y - p1.y;
```

задают приращения координат для этой пары концевых точек. Следовательно, d легко определить, и новое значение pl.y находится путем добавления приращения к старому значению:

```
p1.y += (W. right - p1.x) * dely / delx
```

Аналогичные рассуждения используются при отсечении на остальных трех границах окна.

3. Как определить информацию о памяти (размер ОЗУ ...)

Самый простой способ - использовать следующую функцию

```
Function GetRAMSize:integer;
var MS : TMemoryStatus;
Begin
 GlobalMemoryStatus(MS);
 Result := MS.dwTotalPhys;
end;
```

Функция возвращает размер ОЗУ в байтах. В общем функция GlobalMemoryStatus заполняет структуру типа TMemoryStatus, которая имеет ряд достаточно полезных полей:

dwTotalPhys	Полный объем ОЗУ (т.е. физической памяти)
dwAvailPhys	Свободный объем ОЗУ (как правило небольшая величина)
dwTotalVirtual	Полный объем виртуальной памяти
dwAvailVirtual	Свободный объем виртуальной памяти
dwMemoryLoad	Процент использования памяти (0-не используется, 100-используется вся)
dwTotalPageFile	Общий размер данных (в байтах), которые могут быть сохранены в файле подкачки (но это не является его размером на диске !!)
dwAvailPageFile	Доступный объем в файле подкачки

Прим. Перевод названий корявый - подробности в win32.hlp :))